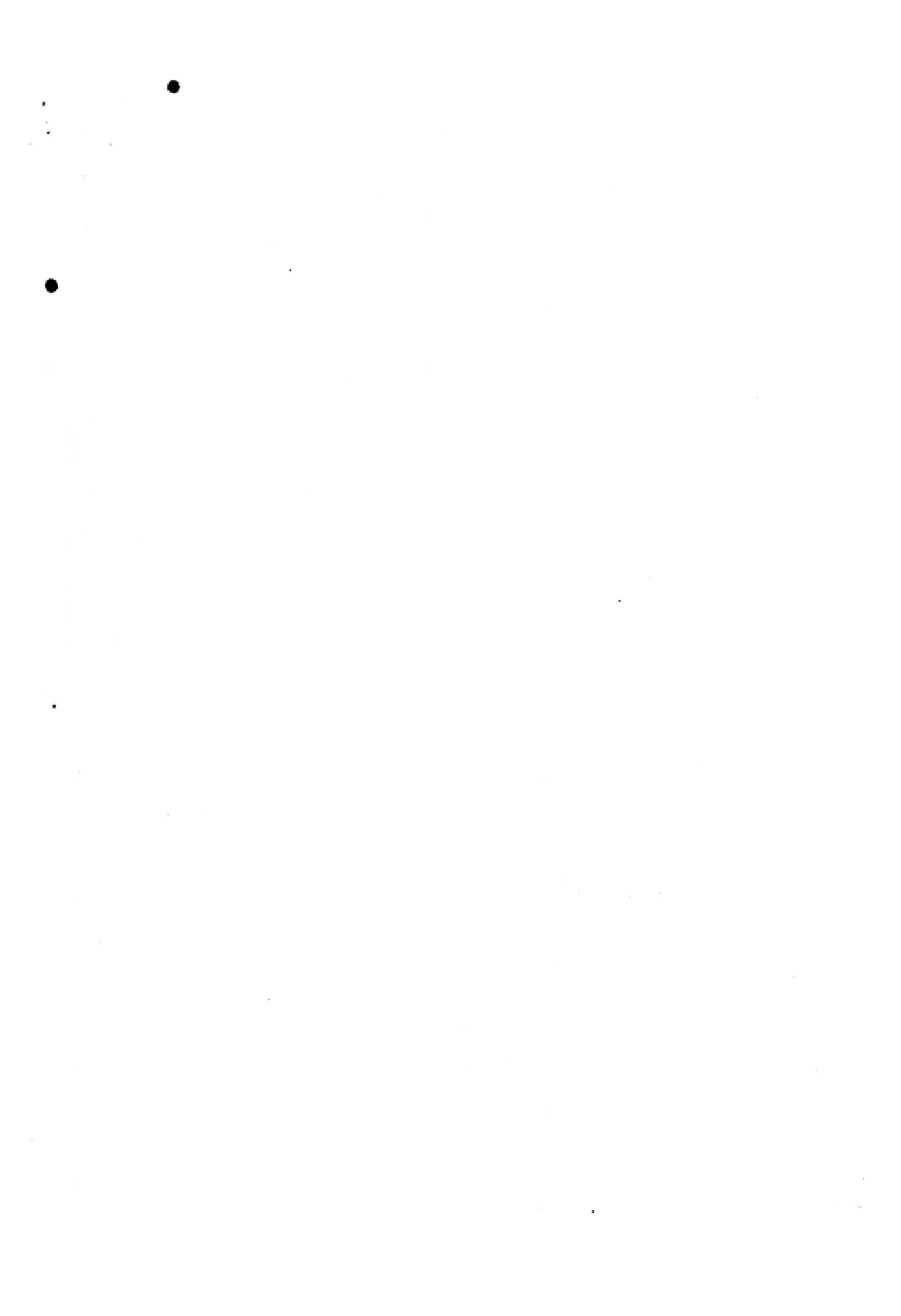
NEC パーソナルコンピュータ PC-8001
N-BASICプログラミング教本

廣済堂出版

——プログラムを組む人のための——

パーソナルコンピュータ

NEC PC-8001

N-BASIC プログラミング教本

横山淳

プログラムを組む人のための

NEC-PC8001

N-BASICプログラミング教本

横 山 淳

廣済堂出版

まえがき

ここ2，3年の間にパーソナルコンピュータが急速に発展してきました．そしてこれを自分のものとして活用したいという人が増え，また学校教育や企業の実務などに利用しようという傾向も強まりつつあります．このようなパーソナルコンピュータを使おうとする人々のために，BASICプログラミングについて解説することが，本書のねらいです．

パーソナルコンピュータにもいろいろな機種がありますが，本書で解説の対象とするのはNECのPC-8001というコンピュータであり，そこで用いられるN-BASICというプログラム言語です．

本書は初心者でも理解できるように，予備知識を前提とせず，第一歩からできるだけ平易に説明するように努めました．したがって初めてコンピュータに接する独習者用として，あるいは講習会や学校教育などにおけるテキスト・参考書として利用していただけるものと思います．

本書は13の章から構成されています．第1，2章は総論的な部分であり，N-BASICプログラミングの入口です．第3章から第11章までは各論であり，N-BASICに含まれるほとんどの命令について，文法的な説明を中心にして記述してあります．第12，13章ではプログラム作成における考え方や応用例を掲げてあります．また各章末に簡単な理解度テスト問題を付けてあります．

記述内容については誤りのないように努めましたが，筆者の微力のため，そのおそれがまったくないとはいいきれません．お気付きの点は御教示いただければ幸甚です．

本書の作成に際しては，NEC Bit-INN横浜および日本マイクロコンピュータ株式会社から甚大な御協力をいただきました．また印刷・出版については広済堂出版社にたいへんお世話になりました．ここに厚く御礼申し上げます．

プログラミングの学習に本書が役立ち，パーソナルコンピュータ活用の一助になれば幸いです．

1980年5月

著　者

目次◉PC−8001 N−BASICプログラミング教本

第1章　はじめに

第2章　プログラミングの基礎

第3章　データの入力

第4章　処理の流れの制御

第5章　数値データの演算

第6章　文字データの処理

第7章　配列による処理

第8章　画面への出力

第9章　プリンタによる印字

第10章　カセットテープの使い方

第11章　ディスクファイルの使い方

第12章　プログラムの組み方

第13章　応用プログラム作成の手引

第1章　はじめに

コンピュータにはじめて接する方は，そもそもコンピュータとは何かについて，まだほとんど知らないわけです．そこで話の出発点として，まず本章の1·1で，コンピュータとはどんなものか，プログラムとは，BASICとはいったい何か，ということを解説します．

次に1·2では，本書で解説の対象とするPC－8001というコンピュータについて，そのあらましを紹介します．

そしてその機械の操作方法を1·3で説明しておきます．

1·1　コンピュータとBASIC

ENIAC（エニアック）という名前で呼ばれる機械が1946年に出現しました．これが世界で最初の電子計算機であるといわれています．すなわち現在のコンピュータの元祖にあたるわけです．このENIACはペンシルバニア大学の研究メンバーによって開発されたもので，18800本の真空管を組込んだ大仕掛けな機械でした．現在のコンピュータと比べればまだ未熟なものですが，当時としては驚きの眼で見られたことでしょう．

その後，20世紀後半に入ってコンピュータは実用化の時代を迎え，今日までの30余年の間に急速な進歩をとげ，あらゆる分野で広く利用されるようになりました．人類の長い歴史から見ればコンピュータの出現はつい最近のことだといえますが，その発展ぶりはまったく驚異的です．

ところで，コンピュータを使う場合にはいくつかの問題点があります．そのひとつは，コンピュータは非常に高価な機械だということです．数百

万円から数億円という金額は，企業・官庁・学校などコンピュータを利用しようとする組織体にとって大きな経済的負担です．ましてや個人でコンピュータを買うなどということは，とてもできない相談です．

またもうひとつの問題点は，コンピュータを使うには専門的な知識・技術が必要だということです．1人前のコンピュータ技術者になるには，1～2年間の教育を受けた後さらに実務経験を積まねばなりません．誰でもすぐに，というわけにはいきません．

さらにまた，コンピュータを使おうとするときは一定の事務的な手続きを経て処理を依頼し，結果が出るのを待つことになるので，今すぐ目の前で結果を欲しいと思ってもなかなかそうはいきません．

これらのことは，「コンピュータは専門家の独占物である」という意識を生んだり，あるいはコンピュータに対するアレルギー反応を生んだりするという結果をもたらします．ところが最近になって，マイクロコンピュータの出現により状況が変わってきました．

マイクロコンピュータ(略して，マイコン)の出現をもたらしたのは半導体エレクトロニクスの技術の進歩です．ENIACのような初期のコンピュータの電子回路に用いられた真空管は現在では姿を消し，トランジスタやダイオードからIC(集積回路）に変わり，さらに高密度なLSI（大規模集積回路）へと進歩してきました．そしてコンピュータの中心部分の複雑な回路を1個ないし数個のLSIにまとめたものが出現したのです．これがマイクロコンピュータ（あるいはマイクロプロセッサ）と呼ばれるものです．

このLSIを中心として，それにキーボードを取付け，さらにディスプレイ装置やカセットテープ装置その他の周辺機器を接続して，ひとつの独立したシステムとして使えるように構成したものが，1977年ごろに登場しました．これもマイクロコンピュータと呼ばれますが，小さいながらも1人前のコンピュータということができます．個人でも買える程度の価格で各種のものが相次いで市販され，人々の注目を集めて急激に広まってきました．そしてこのような形態のマイコンは，最近ではパーソナルコンピュータと呼ばれることが多くなりました．

「パーソナル」というのは「個人用」ということです．すなわちわれわれ1人1人が自分の持ち物として身近に置き，使いたいときに自由に使いこなすのがパーソナルコンピュータだというわけです．ENIAC以来30余年にして，われわれの手もとにコンピュータという道具を持つことができるようになったのです．

パーソナルコンピュータの大きな特徴として，次の3つをあげることができるでしょう．

①低価格である.──→個人でも買える.

②小型である.──→持ち運びが簡単で，どこでも使える.

③使い方が容易である.──→誰でも使える.

さてこうなるとパーソナルコンピュータは良いことづくめで，大型コンピュータはもう不要なのだと思われるかも知れませんが，そうではないのです．非常に大量のデータの処理や，大規模で複雑な超高速の処理などにはパーソナルコンピュータは不向きです．そのような仕事は大型コンピュータにまかせて，上にあげた3つの特徴に合った使い方を1人1人が考え，自分の目的のための用具として使いこなすことが，パーソナルコンピュータを生かす道でしょう．

パーソナルコンピュータの利用範囲はいわば無限ともいえますが，例えば次のようなものがあります．

① ホビー用：趣味としてゲームやプログラムの創作を楽しむ.

② 教育用：コンピュータを理解させるための教材として，あるいはプログラミング演習用として，さらに数学・統計学その他の教科で

の教育手段として，学習に興味を持たせ効果を高める.

③ ビジネス用：販売・在庫・経理・給与計算その他の業務処理のために，あるいはビジネスデータの分析やシミュレーションによって意思決定のための用具として活用する.

④ 科学技術計算用：研究者・技術者が日常の計算のために手もとに置いて利用する．あるいは計測器と接続してデータの収集・解析に用いる.

⑤ 家庭用：ホームコンピュータとして家事に関するさまざまなデータ処理のために，あるいは子供の算数や英語などの勉強用具として活用する.

⑦ 端末用：他のコンピュータと接続して安価なターミナルとして用いる.

その他，使用者の創意くふう次第でさまざまな使い方があるでしょう.

今まで，ENIACから始まってパーソナルコンピュータまで話をしてきました．ところで,「コンピュータ」とはそもそもどんな機械なのでしょうか．詳しくいえばいろいろありますが，簡単に表現すると次のようにいうことができるでしょう.

「コンピュータは，命令されたとおりに仕事(データ処理作業）をする機械である.」

命令するのは人間です．電源を入れても，命令しなければコンピュータは何もしません．コンピュータが自発的に仕事を探してきてそれを行なう，というようなことはないのです．しかし，命令されたことは非常に速く忠実に（バカ正直に!）実行します．だから，うまく使えば有能な働き者として役に立ちます.

では，人間はコンピュータに向って，どのようにして命令するのでしょうか．たとえ相手が機械でも，いろいろなことを命令するには言語（ことば）が必要です．しかしコンピュータは，日本語や英語など（人間語）は理解できません．ではどうするのかというと，コンピュータ用の特別な言語があるのです．この特別な言語を理解できるようにコンピュータは作られているのです.

コンピュータ用言語にもいろいろな種類があります．大型コンピュータではCOBOL，FORTRAN，PL／1などの言語が使われていますが，パーソナルコンピュータではBASIC（ベーシック）という言語が一般に使われています．BASIC はもともとTSS（タイム・シェアリング・システム）用としてアメリカのダートマス大学で開発されたものですが，現在ではパーソナルコンピュータ用言語の主流となっています.

コンピュータ用言語もことばですから，いろいろな単語があり，文法があります．文法の規則に従って単語を並べると命令文ができます．ですから，パーソナルコンピュータを使うには BACIC という単語の文法を覚えることが必要です．つまり，新しい外国語をこれから学習するようなものです．しかし恐れる必要はありません．なぜなら，BASICは英語やフランス語などよりもはるかにやさしいからです．ただ注意すべきことは，コンピュータ用語は非常に「文法にうるさい」ということです．ちょっとでも文法の規則に違反した表現をすると，コンピュータは「命令がおかしい!」といって仕事をしてくれません.

さて，コンピュータとは命令どおりに働く機械だということを述べましたが，それだけではなく，次のような能力を持った機械です．これはコンピュータを特徴づける非常に重要な能力です.

「コンピュータは，一連の作業を続けて自動的に実行できる.」

すなわち，コンピュータはいわゆる電卓とは違い，自動的に作業を進めることができるのです．ただしそのためには「プログラム」というものが必要です．プログラムとは，一連の作業の内容をいくつかの命令文を並べて表現したものです．つまり命令文の集まりであり，いわば「作業命令書」です.

コンピュータに自動的に仕事をさせるには，ま

ず人間が作業の手順計画を考えてプログラムを作り，それをコンピュータの中に入れて記憶させるのです．それから「実行せよ」と命令すれば，プログラムに並んでいる命令文に従って自動的に作業を進めてくれるのです．

プログラムを作ることをプログラミングといいますが，コンピュータを使おうとする人にとってたいへん重要な仕事です．

以上にコンピュータとかBASICとかプログラムなどについて，それがどんなものであるかのあらましを述べてきました．これを出発点として次のことを学習すれば，あなたは自分でパーソナルコンピュータを使えるようになります．前述のことを頭に入れて置けば，それ以外の予備知識は不要です．

① 操作方法（主としてキーボードの操作）

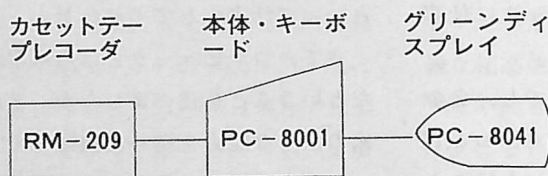
② プログラムの作り方（BASIC言語を用いる）

これらについての詳細は，次節以降さらに各章において逐次解説します．

1·2 PC－8001のあらまし

現在市販されているパーソナルコンピュータにはいろいろな種類があります．機種が異なれば，機械の構造や装置構成あるいは操作方法も多少異なります．また使われるBASIC言語の文法も機種によって若干の相違があります．本書で解説の対象とするのは，「PC－8001」というパーソナルコンピュータとそこで使われるBASIC言語です．本節ではPC－8001を紹介し，そのあらましを説明します．

PC－8001はNEC（日本電気株式会社）の製品であり，現在のパーソナルコンピュータの代表的な機種のひとつです．PC－8001で使われるBASIC言語は「N－BASIC」と呼ばれます．これは，MICROSOFT™ BASICの言語仕様を基にして，さらに機能を追加拡張したものです．

PC－8001には各種の周辺機器を接続することができますが，そのもっとも基本的なシステム構成は次のとおりです．

システム構成の中心はPC－8001であり，この中には演算・記憶・制御の機能を持つ装置がLSIの形で内蔵されています．人間に譬えれば頭脳に相当します．キーボードは，データや命令などをコンピュータの中へ入れるための装置(入力装置)です．ディスプレイ装置は，処理結果を目で見えるようにコンピュータの外へ出すための装置（出力装置）です．カセットテープレコーダは，データやプログラムを本体の外へ取り出して別に保存するための装置（外部記憶装置）です．

一般にコンピュータは（大型でも小型でも），演算・記憶・制御・入力・出力の５大機能で構成されています．PC－8001を中心とする上掲のシステム構成は，小さいながらもこの５大機能を備えた独立したコンピュータシステムであるといえるわけです．

以上のほかにさらに，プリンタ・ミニディスク装置・カラーディスプレイなどが接続でき，また，拡張ユニット（PC－8011）やケーブルユニット（PC－8062）などを介して，他の計測器やコン

ピュータなどと接続することもできます．

次にPC−8001の動作モードについて簡単に述べておきます．動作モードというのは，動作の様式，つまりコンピュータが仕事を行なうときのやり方のことです．PC−8001には次の3つの動作モードがあります．

① 直接モード

② 間接モード

③ ターミナルモード

それぞれのモードでの仕事のやり方のあらましを図に示しておきます．

まず電源を入れると，ディスプレイ装置の画面にOKという文字が現われます．これは，コンピュータが現在「手空きの状態」であって，「指示を受け入れる準備ができている」ということを表わしています．この状態をコマンドレベルといいます．

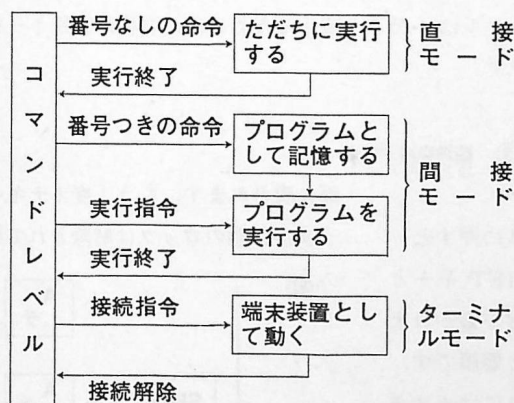
そこでいろいろな指示を与えるわけですが，その与え方によって動作モードが分かれます．

直接モードでは，コンピュータが1個の命令を受け取ると，ただちにそれを実行します．実行が終わればまたコマンドレベルに戻ります．実行ずみの命令はコンピュータの記憶装置には残りません．なお，このモードで動作させるときは，命令に番号をつけません．直接モードでは，電卓と同じような使い方ができます．

頭に番号のついた命令文を与えると，コンピュータは間接モードであると判断します．このときはそれをすぐには実行せず，記憶装置に入れて記憶します．ですから複数個の命令文を次々と与えて記憶させることができます．この命令文の集まりがプログラムなのです．

記憶されたプログラムは実行指令によって実行を開始し，命令文につけられた番号の順に自動的に仕事が進行します．プログラムの実行が終われぱコマンドレベルに戻ります．実行が終わっても，記憶装置内のプログラムは消えずに残っています．したがって，同じプログラムを何回でも実行させることができます．この点が直接モードと異なります．

ターミナルモードは，PC−8001を他のコンピュータに接続して端末装置として使うときのモードです．TERMという指令を与えるとこのモードになります．

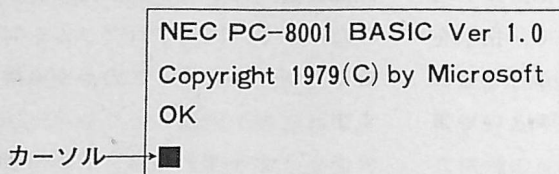
以上3つのモードのうちで重要なのは，間接モードによるプログラムの実行です．これについては第2章でさらに説明を加えます．

1·3　キーボードの操作法

キーボードは，データや命令などをコンピュータの中へ入れる（入力する）ための装置です．キーを押して入力することをKey-inといいます．本節ではキーボードの操作法について説明します．

キーボードは左半分と右半分とに分かれて多数のキーが並んでいます．左側の部分は普通のタイプライタとよく似たキー配列になっています．右側の部分は数字を中心とした配列になっており，テンキーと呼ばれます．

また，キーを２大別すると，普通キーと特殊キーとがあります．濃い色のキーが特殊キーです．

PC−8001およびディスプレイ装置の電源を入れると，画面に次のような表示が現われます．

```
NEC PC-8001 BASIC Ver 1.0
Copyright 1979(C) by Microsoft
OK
■  ←カーソル
```

なお画面の下部にはファンクションキーの表示が出ますが，これについては後で説明することにします．

OKという文字の下に四角いものが点滅していますが，これを「カーソル」と呼びます．一般に，普通キーを押すと，その文字が画面のカーソルのある位置に表示されます．そしてカーソルは自動的に右へ１桁移動します．つまりカーソルは，「次に表示する文字」の位置を示すものです．

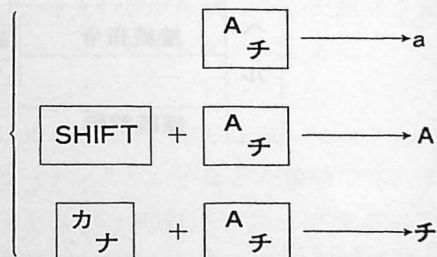
１個のキーを押してすぐに手を離せば，１個の文字が画面に書かれます．しかし，手を離さずに約１秒以上押していると，手を離すまでは同一文字が連続して表示されます．このようなキーの働きを「リピート機能」と呼びます．

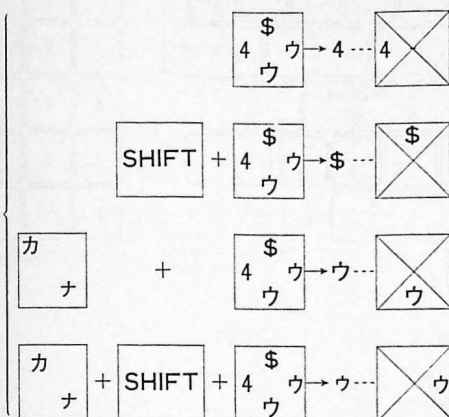
さて，１個のキーに２文字ないし４文字が付いているものが多くあります．その中の１文字を指定するために次の３種の特殊キーを使います．

①　SHIFTキー，②　カナキー，③　GRPH キー

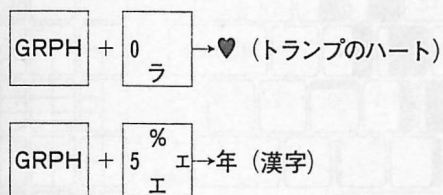
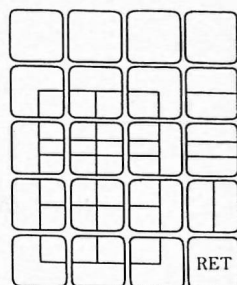
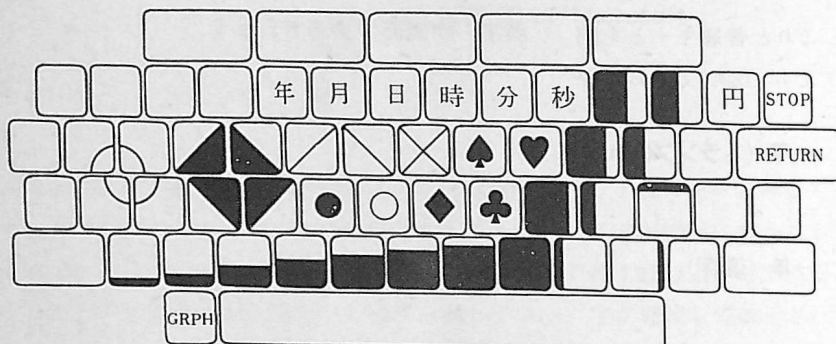
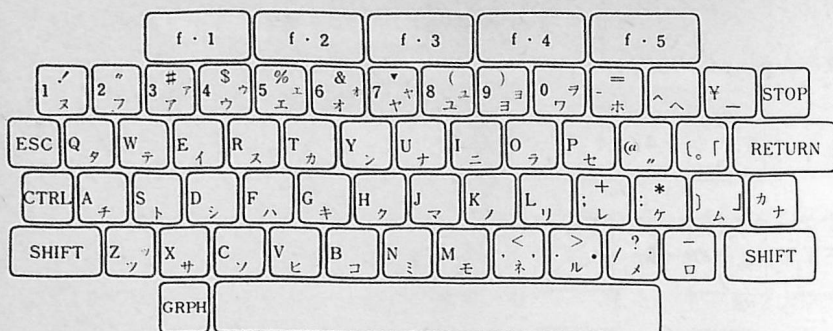
これらの使い方を次に説明します．

まず英文字ですが，１個のキーを単に押すと，英小文字が画面に表示されます．SHIFT キーと英字キーとを同時に押すと，英大文字が書かれます．つまり普通のタイプライタと同じ要領です．

次にカナ文字の書き方ですが，これにはカナキーを使います．カナキーは１度押すとキーが凹んでブロックされます．つまりカナ文字状態に固定されます．この状態でキーを押すとカナ文字が画面に現われます．もう１度カナキーを押せば，カナ文字状態のロックは解除されて普通の状態に戻

ります．

また，1個のキーに4文字も付いているものは，次の例のようになります．

次にGRPHキーですが，これと普通キーとを同時に押すと，グラフィック文字（図形文字および漢字）が画面に表示されます．

〔例〕 GRPH ＋ 0 ラ →♥（トランプのハート）

GRPH ＋ % 5 エ エ →年（漢字）

グラフィック文字はキーには書かれていないので，その位置は次図を参照してください．

キーボードの右半分つまりテンキーの最上段にある４個のキーは，画面制御キーと呼ばれます．これについて次に説明します．４個とも，単に押せば下側，SHIFTキーと併用すれば上側の働きが指定されます．

HOME CLR　このキーを単に押すと，画面に書かれていた文字がすべて消えます．これを「画面消去」あるいは「画面のクリア」といいます．ただし，画面の最下部にあるファンクションキーの表示は消えません．なおこのとき，カーソルは画面の左上隅に行きます．

また，このキーとSHIFTキーとを同時に押すと，画面は消えずに，カーソルだけが左上隅に行きます．なお，画面の左上隅の位置を「ホームポジション」と呼びます．

↓↑　→←　これらのキーを押すと，矢印の方向へカーソルが１桁ずつ移動します．画面に書かれている文字の上をカーソルが通過しても，文字は消えません．リピート機能も使えます．

なお，カーソルが画面の右端にあるときに→を押すと，カーソルは次の行の左端に行きます．カーソルが画面の左端にあるときに←を押すと，カーソルは１行上の右端に行きます．カーソルが画面の最上行にあるときに↑を押しても，画面の状態は変化しません．また，カーソルが面面の最下行にあるときに↓を押すと，画面全体が上に移動し，最上行にあった文字は画面から消えます．こ

れを「スクロール」と呼びます．

INS DEL　このキーを単に押すと，カーソルの左側の文字が消されます．また，このキーとSHIFTキーとを同時に押すと，カーソルから右の文字が右に移動して，そこに空白ができます．画面に文字を書きまちがえたときには，このキーを使って不要な文字を削除したり，空白を作ってそこに文字を挿入したりすることができます．

以上が画面制御キーの働きです．

次に，STOPキーとESCキーですが，これは実行を停止させるときに使います．第4章（4·7）を参照してください．

次にCTRLキーについて説明しておきます．このキーは「コントロールキー」と呼ばれ，他のキーと同時に押すと特定の働きをします．それを下に示しておきます．ここで©というのはCTRLキーの意味です．

©＋Cキー →プログラムの実行中（入力待ちのとき）にこれを押すと，実行を停止してコマンドレベルに戻ります．

©＋Lキー →画面をクリアする．

©＋Kキー →カーソルをホームポジションに戻す．

©＋Hキー →1文字を削除する．

©＋Rキー →1文字を挿入する．

©＋Nキー →カーソルを1項目ごとに右へずらす．

©＋Bキー →カーソルを1項目ごとに左へずらす．

©＋Eキー →カーソル位置から後を消去する．

©＋Jキー →カーソルから右の部分を，次の行にずらす．

©＋Iキー →タブ（8桁ごと）

©＋Gキー →ブザーを鳴らす．

次にRETURNキーですが，これは非常に重要な働きをします．RETURNキーを押すと，そのとき画面上でカーソルがある行が，命令またはデータとしてコンピュータの中へ入って行きます．RETURNキーを押さなければ，画面上の文字はそこに表示されているに過ぎません．なお，テンキーの右下隅にあるRETキーもRETURNキーと同じものです．

最後にファンクションキーについて説明しておきます．

ファンクションキーとは，キーボードの最上段に並んでいる5個のキー（f・1～f・5）です．これらのキーには，それぞれ15字以内の文字を記憶させることができます．1個のファンクションキーを押せば，そこに記憶されている文字が画面に表示されます．つまり複数個のキーを押したのと同じ結果になるわけで，Key-in作業をスピードアップすることができます．

ファンクションキーは5個ですが，これとSHIFTキーとを併用することにより10種類のキーとして使うことができます．この10種類を，1番から10番までの番号で呼びます．番号は次のようにつけられています．

〔例〕 f·1 →1番，　SHIFT ＋ f·1 →6番，

f·5 →5番，　SHIFT ＋ f·5 →10番，

電源投入時には，各ファンクションキーには一定の文字がすでに記憶されており，画面の最下段に1番から5番までの記憶内容が表示されます．SHIFTキーを押せば6番から10番までの内容が表示されます．ただしこのときの表示は，15文字のうちの6字分です．15文字全部を見たいときは次のコマンドを使います．

KEYLIST

……ファンクションキーの内容を画面に表示する．

これを画面に書いてRETURNキーを押すと，次のように画面に表示されます．

```
keylist
HT        time $
auto      key
goto      print
list      list. CR ↑ ↑
run CR    cont  CR
Ok
```

ファンクションキーの内容を使用者が定義することができます．次のコマンドを使います．

KEY〈ファンクションキーの番号〉，"〈文字列〉"

定義したい文字列は引用符（"）で囲みます．あるいは次の例のように，CHR$関数を使うこともできます．

〔例〕　KEY 5，"run"+CHR$ (13)

CHR$については第6章（6·1）を参照してください．13はRETURNキーに対応する文字コードです．

ファンクションキーはデータを入力するときにも利用できます．第3章（3·3）を参照してください．

キーボードの操作法の説明は以上で終わりますが，自分の手でキーに触れて慣れることが大切です．

ここで，直接モードによる計算の方法（電卓的な使い方）を述べておきます．画面に次のように書いてRETURNキーを押せば，そのすぐ下に計算結果が表示されます．

?〈計算式〉

〔例〕？　5＋4＊3－2 ──→答は15になります．

　　　？　(5＋4)＊3－SQR (16) ──→答は23になります．

＊は乗算の記号です．SQR(16)は$\sqrt{16}$の意味です．これらの演算式の書き方の詳細は，第5章で

解説します．なお，？のかわりにPRINTと書いて　　も同じ意味になります．

理解度テスト

(1)　電卓的な使い方（直接モード）で，次の計算をしなさい．

①　2×3.14159×100

②　$\sqrt{2}+\sqrt{3}$

③　2×2＋3×3＋4×4＋5×5

④　$\dfrac{123+456}{789-1000}$

⑤　$\sqrt{3\times3+4\times4}$

(2)　次の文字を，指定した番号のファンクションキーに定義しなさい．

①　あなたの氏名（2番）

②　あなたの電話番号（5番）

③　3.14159265359（7番）

④　INPUT（10番）

⑤　NEXT（9番）

第2章　プログラミングの基礎

前章で述べたように，コンピュータを活用するにはプログラムが必要です．プログラムを作りコンピュータに入れて記憶させ，それを実行させれば，自動的に処理が行なわれて処理結果を得ることができます．

プログラムによる処理の仕方やBASIC 文法などの詳細は次章以降で解説しますが，それに先立って，プログラミング全般に共通する基礎的な事項を本章で述べておきます．

まず2·1では，プログラムとはどんなものかについてさらに具体的に解説します。

次に2·2では，プログラムの書き方についての一般的な規則を示します．

2·3では，プログラムの作成から実行・保存にいたるまでの各ステップについて，その方法を説明します．

また2·4では，いくつかのプログラム例を掲げて，その処理形態や基礎的な命令文を紹介します．

2·1　プログラムとは

プログラムは「命令文の集まり」である，ということはすでに述べました．BASIC では命令文のことをステートメントといいます．そしてプログラムを構成するステートメントには必ず番号をつけることにきまっています．この番号を「行番号」と呼びます．

行番号のついていない命令文はプログラムの構成要素とはみなされず，直接モードでただちに実行されてしまいます．(1·2および1·3参照)

行番号のついた命令文は，すぐには実行されないで，いったん記憶装置に記憶されます．それから「実行せよ」という指令（このような指令はコマンドと呼ばれる）を与えると，コンピュータは，原則として，行番号の若いステートメントから順に自動的に実行します．ですから，行番号は実行順序を示す重要なものです．

プログラムを構成する「行番号のついたステートメント」のそれぞれを「行」と呼びます．

まずとにかく，プログラムの例を示しましょう．

〔例〕 単価50円の品物が何個かある．その数量をコンピュータに入れて，金額（＝単価×数量）を計算させる．計算結果（金額）は画面に表示させる．

```
10  INPUT  S
20  K=50 * S
30  PRINT  K
40  END
```

これは非常に簡単なプログラムです．4つのステートメントが並んでおり，10番から40番までの行番号がついています．ちょっと英語と似ていますが英語ではありません．これがBASICという言語で書かれたプログラムなのです．

ではこのプログラムを実際にコンピュータに入れて実行してみましょう．プログラムの入れ方や実行のさせ方などは2·3に説明してあるので，わからなかったらそれを読みながら，とにかく次のようにやってみてください．

① プログラムをコンピュータに入れて記憶させます．

プログラムは1行分ずつ入れます．キーボードのキーを押して画面に1行分を書き，それからRETURNキーを押してください．そうすると，その1行がコンピュータの中に入って行き，記憶装置内に記憶されます．このとき，RETURNキーを押すのを忘れないでください．これを押さないと，中へ入って行きません．

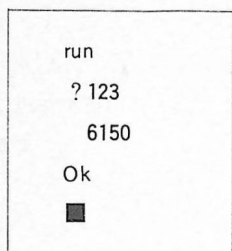
1行分ずつ入れる：　10　INPUT　Sと書いて──→RETURNキーを押す．

以下同様にして，4行分すべてを入れてください．

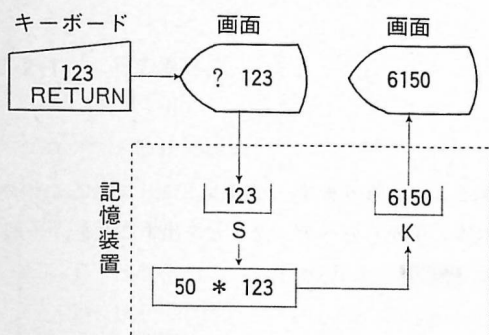
② プログラムを実行させます．

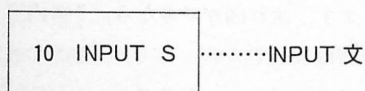
プログラムを入れ終わったならば，画面にRUN

と書いてRETURNキーを押してください.「RUN」というのは「プログラムを実行せよ」という意味の命令（コマンド）です.

実行が始まると画面に疑問符(?)が現われるは

ずです. そうしたら数字キーを押して任意の数量(例えば, 123) を画面に書き, RETURNキーを押します. するとそのすぐ下に, 計算結果 (例えば, 6150) が表示されます. これでプログラムの実行は終わります. このとき画面は図のようになっています.

ここでまた, RUNというコマンドを与えれば,

新たに実行が開始されます.

このプログラムの各行について, 説明を加えておきます. 図を見ながら読んでください.

```
10 INPUT S
```
……INPUT 文

「INPUT」というのは,「データを読み込みなさい」という意味のことばです. コンピュータに演算その他の処理をやらせるためには, データを中へ入れなければなりません. これをコンピュータの側から見れば,「読み込む」ということになります. コンピュータ本体の外から中へ, データあるいは命令などを入れることを, 一般に「入力」といいます.

INPUT という入力命令はキーボードを使って行なわれます. つまり人間がキーを押してデータを入れてやるのです. RUNというコマンドを与えたときに画面に疑問符(?)が現われましたが, これは, コンピュータが人間に対して「データを入れてください」と呼びかけているのです. そこで, キーを押して画面にデータを書き RETURN キーを押せば, それが読み込まれるのです.

読み込まれたデータは, 記憶装置に入って記憶されます. INPUT文に書いてある「S」は, 入力したデータを記憶させる記憶場所の名前です. 記憶装置の中にはたくさんの記憶場所が並んでいますが, その中の適当な場所をコンピュータが選んで, その場所にSという名前をつけてくれるのです. いいかえれば, 人間がプログラムの中に名前を書けば, コンピュータが自動的に記憶場所の割り当てをしてくれるのです.

このような個々のデータの記憶場所を「変数」

と呼び，それにつけた名前を「変数名」といいます．これはプログラムにおいて非常に重要な概念です．

```
20 K=50 * S
```
………代入文

これは代入文と呼ばれるステートメントです．「=」の右側のものは計算をする式です．「*」は乗算を表わす演算記号です．=の左の「K」は変数名です．代入文の=は，左右が「等しい」という意味ではなく，右辺の値を左辺に「代入」せよという意味です．詳しくは次のようになります．

「右辺の式を計算し，その値を左辺の変数（記憶場所）に入れて記憶せよ.」

また，=のすぐ右の「50」は，データの値を直接に表わしています．このようなものを「定数」と呼びます．プログラムの実行中に，変数の値は変わることがありますが，定数の値は変化しません．

```
30 PRINT K
```
………PRINT 文

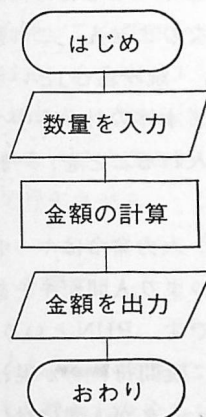
「PRINT」は「画面に表示しなさい」という意味のことばです．Kという名前の変数に記憶されている値が画面に表示され，人間が見ることができるようになります．このように，コンピュータ本体の中から外へデータなどを出すことを，一般に「出力」といいます．

```
40 END
```
………END 文

これは実行の終了を意味します．コンピュータは「Ok」と表示してコマンドレベルに戻ります．

以上のように，プログラムを構成する各ステートメントは行番号に従って順に実行されます．ですからステートメントの並び方の順序が非常に重要です．そこで，プログラムを作るには「どのような作業を，どのような順序で，コンピュータにやらせるべきか」という手順計画を立てねばなりません．

この手順計画を図式化したものを，流れ図とかフローチャートといいます．手順が複雑な場合は，頭の中だけで考えていると混乱してわからなくなったり，まちがったりします．そこで，流れ図を書いて目で見えるようにすると，手順の流れがはっきりしてきます．複雑な長いプログラムを作るときは，まず流れ図をしっかりと書くことをおすすめします．流れ図ができたら，その内容をBASIC言語によってステートメントとして表現すればプログラムができるわけです．

上例のプログラムの流れ図を示しておきます．これは非常に簡単ですが，プログラムの処理の流

れのひとつの基本的な形といえます．それは，

① データを入力する．

② データに対してさまざまな処理を行なう．

③ 処理結果を出力する．

という基本的なステップから成り立っているからです．おおまかにいえば，複雑なプログラムでもおよそこの3ステップから構成されるといえます．

なお，流れ図を書くための記号を2·2·3に掲げておくので，参照してください．

2·2 プログラムの書き方

本節では，プログラムを書く場合の一般的・共通的な規則を示します．それは行の書き方の規則および変数の名前のつけ方の規則です．

また，プログラムを作るときに大切な流れ図の記号も掲げておきます．

2·2·1 行の書き方

プログラムは行の集まりです．行とは，行番号の後にステートメントを書いたものです．行の書き方について次の3つの規則があります．

① 行番号は，0～65529 の範囲内の整数とする．必ず数字で書く．英字などは使えない．

② ひとつの行の長さは，255 字以内とする．行番号や空白も長さに含まれる．

③ コロン(：)で区切れば，ひとつの行の中に複数個のステートメントを書ける．

この③の規則を使うと，前節のプログラムは次のように書くこともできます．

```
10 INPUT S：K=50*S
20 PRINT K
30 END
```

もっと極端に書けば，次のように1行にまとめてしまうことも可能です．

```
10 INPUT S：K=50*S：PRINT K：END
```

2·2·2 名前のつけ方

変数(個々のデータを記憶する場所)は名前（変数名）で表わされます．この名前のつけ方について次の規則があります．これを命名の規則といいます．

① 名前の最初の文字は必ず英字とする. 2字目以降は英字でなくてもよい.
② 名前の中に予約語が含まれてはいけない. また特殊文字が含まれてはいけない.
③ 名前の長さは2文字以内とする.

予約語というのは, BASIC言語の中で特定の意味がすでに定められているものです. 例えば, 前掲のプログラム例の中の, INPUT, PRINT, ENDなどです. 短い予約語の例としては, IF, ON, ORなどがあります.

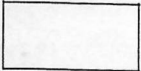
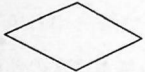
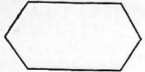
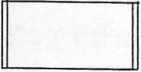
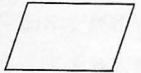
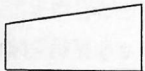
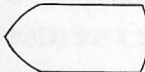
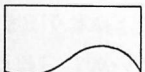
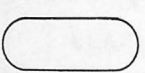
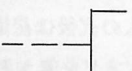
また上記③では, 名前の長さは2字以内としてありますが, 実際には3字以上の名前を書くことができます. ただし, コンピュータは最初の2文字だけを見て変数を区別します.

正しい名前の例：A, AA, AB, A1, A 2
誤った名前の例：1A, A－, A＊, IF, AIF
同一とみなされる名前：AAA, AAB, AA1 →この3つは同じ変数を表わすものとみなされる.

文字型の変数は, 変数名の後にドル記号($)をつけて表わす.

BASICで取扱うデータを大別すると, 数値型と文字型とがあります. 変数名の後に$がついていると, それは文字型変数とみなされます. なお文字型定数は, 原則として引用符(〃)で囲んで表わします.

文字型変数名の例：A$, AA$, AB$, A1$
文字型定数の表わし方の例："TOKYO", "ゴウケイ キンガク"

文字型データの詳細については第6章を参照してください.

$のついていない変数名は数値型ですが, 実は, 数値型データにも3つの型式があります. その詳細は第5章を参照してください.

2·2·3 流れ図の記号

流れ図（フローチャート）を書くための記号はいろいろあります. しかし普通はわずかな種類の記号で間に合います. ここでは,JIS規格(C6270)に規定されている記号のうち主要なものを引用して紹介しておきます. 流れ図を書くときに参考にしてください.

記　　号	意　　味
（長方形）	処理（process） あらゆる種類の処理機能を表わす．
（ひし形）	判断（decision） いくつかの択一的径路のうち，どの径路をとらせるかを決める判断，またはスイッチ式の操作を表わす．
（六角形）	準備（preparation） スイッチの設定，指標レジスタの変更，ルーチンの初期値の設定など，プログラム自身を変えるための命令，または命令群を修飾する機能を表わす．
（両側二重線の長方形）	定義済み処理（predefined process） サブルーチンなど，別の場所で定義されている命令群，またはいくつかの操作からなる命名された処理過程を表わす．
（平行四辺形）	入出力（input/output） 情報を処理可能にする入力機能，または処理済みの情報を記録する出力機能を表わす．
（上辺が傾いた四角形）	手操作入力（manual input） オンラインけん盤，スイッチ，押しボタンなどを手で操作して，処理中に情報を入れる入力機能を表わす．
（表示の記号）	表　示（display） 処理中に，オンライン表示灯，映像表示装置，操作卓の印字機，作図機などを通じて，情報を人間が利用できるように表示する入出力機能を表わす．
（書類の記号）	書　類（document） 書類を媒体とする入出力機能を表わす．
（円）	結合子（connector） 流れ図のほかの場所への出口，あるいはほかの場所からの入口を表わす．
（角の丸い長方形）	端子（terminal, interrupt） 開始，終了，停止，遅延，中断など，流れ図の端子を表わす．
（注釈の記号）	注釈（comment, annotation） 明りょうにするために，説明または注意を加える機能を表わす．

2·3　プログラムの作成・実行・保存

コンピュータはプログラムに従って自動的に仕事をしてくれます．しかし，プログラムを考えて作り，それをコンピュータの中に入れて記憶させたり，「実行せよ」と指示したりすることは，人間がしなければなりません．またその途中で，コンピュータの中にあるプログラムを目で見て確かめたり，プログラムの内容を修正したり，あるいはプログラムをカセットテープに記録して保存したりするといったことも必要になります．

これらのことは，コンピュータを利用しようとする者が頻繁に行なわねばならない仕事です．本節ではこれらの方法について説明します．方法そのものはいずれも簡単なものですが，ただ説明を読むだけでなく，自分の手で実際にやってみて慣

れることが大切です.

2·3·1　プログラムを入れる

プログラムをコンピュータの中に入れて記憶させる方法は，2·1で簡単に述べましたが，ここでさらに説明を加えます.

プログラムの入れ方の基本は次のとおりです.

「1行分ずつ，画面に書いて，RETURNキーを押す.」

RETURNキーが押されるとコンピュータは，そのときカーソル(■)のある行を見て，その行の先頭に行番号がついていれば，プログラムを構成する行であると解釈して記憶装置に入れます.

入れようとする1行分が長くて，画面の右端まで来てもまだ書ききれない場合は，そこでRETURNキーを押さずに，続けてどんどん書けばよろしい．そして書き終わった所でRETURNキーを押すのです．そうすれば，画面上では2行以上にまたがっていても，それを1個の行とみなして記憶してくれます.

1行分を画面に書くとき，英字は小文字で書けばよいのです.（大文字で書いても差しつかえはありません.）プログラムの行はコンピュータが自動的に大文字に変換して記憶するようになっています．ただし，引用符(″)で囲まれた部分（文字定数）およびREM文の中の文字は，小文字はそのまま小文字として記憶されます.

なお，画面に行を書くとき，ファンクションキーを利用することもできます.（1·3参照）

また，文字を書きまちがえたときは，1·3で述べたようなキーの操作によって修正することができます．あるいは，まちがえたままの状態でRETURNキーを押し，その下にもう一度新たに正しく書いて入れてもよろしい.

次にNEWというコマンドを紹介しておきます.

| NEW |　………記憶されているプログラムをすべて消す.

画面にNEWと書いてRETURNキーを押します．すると，記憶装置の中に記憶されていたプログラムおよびデータはすべて消えてしまいます．新しいプログラムを入れ始めるときは，すでに記憶されている使用ずみのプログラムを消すことが必要です．そうしないと，新旧の両プログラムが混合してしまいます．そこでNEWを使うわけです.

なお，コンピュータの電源を切ると記憶装置内のプログラムやデータはすべて消えてしまいます．ということは，電源投入の直後は記憶装置の中は空ですから，NEWコマンドを使う必要はないわけです.

また，ＰＣ－8001の後部にあるリセットスイッチを押しても，電源投入直後と同じ初期状態になります.

次にAUTOというコマンドを紹介します.

| AUTO　〈行番号〉，〈増分〉 |　………行番号を自動的に発生する.

画面にこのコマンドを書いてRETURNキーを押すと，指定に従って自動的に行番号を発生し画面に表示してくれます．プログラムを入れるときに利用すると便利です．STOPキーを押せばA

UTOの働きは止まります．

行番号や増分の指定の仕方は次の例を見てください．

〔例〕 AUTO　100，50 ──→ 100，150，200，…のように行番号を発生する．
　　　AUTO　10，10 ──→ 10，20，30，40，…のように行番号を発生する．
　　　AUTO　（指定なし）──→ 10，20，30，40，…となる．（上例と同じ）

2·3·2　プログラムを見る

記憶装置内に入っているプログラムを画面に表示して見ることができます．それには次のコマンドを使います．

LIST　〈行番号の指定〉

………プログラムを画面に表示する．

行番号の指定の仕方によって，プログラムの全部または一部分が表示されます．指定の仕方は次の例を見てください．指定の中の記号(－)は，コンマ(，)を使ってもよろしい．

〔例〕 LIST　（番号指定なし）──→ プログラムの最初から最後までをすべてリストする．
　　　LIST　500－700 ──→ 500番から700番までの行をリストする．
　　　LIST　－400 ──→ 最初の行から400番まで．
　　　LIST　400－ ──→ 400番の行から最後まで．
　　　LIST　200 ──→ 200番の行のみをリストする．

上記とよく似た形の次のコマンドがあります．

LLIST　〈行番号の指定〉

………プログラムをプリンタで印字する．

行番号の指定の仕方は上記のLISTコマンドと同じです．プログラムをプリンタ用紙に印字しておけば，いつでも取り出して見ることができるし，記録として保存できます．

2·3·3　プログラムを実行させる

記憶装置内に入っているプログラムを実行させるには，次のコマンドを使います．

RUN　〈行番号〉

………プログラムの実行を開始する．

行番号を指定しないと，プログラムの最初の行から実行を開始します．行番号を指定すれば，その行から実行を開始します．

〔例〕 RUN　（行番号指定なし）──→ もっとも若い番号の行から実行する．
　　　RUN 500 ──→ 500番の行から実行する．

RUNコマンドによってプログラムの実行が開始されるときは，記憶装置内のすべての変数（データの記憶場所）は消去（クリア）されます．数値変数の値は0（ゼロ）になり，文字変数の値は空（何も文字が記憶されてない状態）になります．

なお，実行中のプログラムを途中で強制的に停止させようとするときは，STOPキーを押します．プログラムの停止や再開については，第4章（4·7）でさらに説明します．

2·3·4　プログラムを修正する

プログラムを作り，コンピュータに入れて実行させてみると，予想どおりにならないことがあります．実行の途中でエラーメッセージが現われて止まってしまったり，あるいは計算結果が正しく出てこない，といった事態は誰でもよく経験することです．プログラムの中に誤りや不充分な点があると，このような事態が起こります．

そこで，プログラムの内容をよく調べて誤りの原因を見つけ，それを修正しなければなりません．このようなプログラムのまちがい直しのことを「デバッグ」といいます．デバッグが終わればプログラムは完成したことになります．

あるいはまた，すでに完成しているプログラムの内容を修正して，さらによいプログラムにするという場合もしばしばあります．

そこで本項では，プログラムの修正の仕方を説明します．修正の方法としては，行の追加·削除·訂正および行番号のつけかえなどがあります．

(1)　行の追加挿入

プログラムに行を追加する方法は，2·3·1で述べたプログラムの入れ方と同じです．すなわち，追加したい行を1行分ずつ画面に書いてRETURNキーを押せばよろしい．そうすれば，その行が自動的に該当する行番号の位置に挿入されます．プログラムを書くときに行番号を連続させずに間隔をあけて書くのは，あとから行を追加挿入するのに便利だからです．

(2)　行の削除

プログラムの中の行を削除するには次のようにします．

「削除したい行の行番号のみを画面に書いて，RETURNキーを押す．」

また，何行もまとめて削除するときは，次のDELETEコマンドを使うとらくです．

DELETE　〈行番号の指定〉

………指定された範囲の行を削除する．

行番号の指定の仕方は，2·3·2で説明したLISTコマンドと同じです．

〔例〕 DELETE　150－400 ──→ 150番から400番までの範囲のすべての行を削除します．

(3)　行の内容の訂正

プログラムの中の行の内容を訂正するには，その行番号と新しい行内容とを画面に書いて，RETURNキーを押せばよろしい．そうすると，古い行内容は自動的に消えて，新しい内容が入ります．

あるいは，新しい行を画面に書くかわりに，LISTコマンドによって画面にプログラムを表示して，画面操作用のキーを使って行内容を訂正編集してから，RETURNキーを押してもよろしい．

(4)　行番号のつけかえ

次のRENUMコマンドを使えば，プログラムの行番号をつけかえることができます．

RENUM　〈新番号〉，〈旧番号〉，〈増分〉………行番号をつけかえる．

ここで新番号というのは，新しくつける行番号の最初の番号です．もしこれの指定を省略すると，自動的に新番号は10であるとみなされます．

旧番号というのは，番号のつけかえを始める現在のプログラムの行番号です．もしこれの指定を省略すると，現在のプログラムの最初の行からつけかえが始まります．

増分というのは，新しくつける各行番号の間隔です．もしこれの指定を省略すると，自動的に増分は10であるとみなされます．

これらの指定の仕方の例を次に掲げます．

〔例 1〕 RENUM （すべての指定を省略）

この場合は，現在のプログラムの最初の行から最後の行までのすべての行番号をつけかえます．新しい最初の行番号は10となり，それ以降は10ずつ増加した行番号となります．

〔例 2〕 RENUM 100，，50 （旧番号の指定を省略）

この場合も，現在のプログラムの最初から最後までのすべての行番号をつけかえます．新しい最初の行番号は 100 となり，それ以降は50ずつ増加した行番号となります．

〔例 3〕 RENUM 1000，950，100

現在のプログラムの 950 番から以降の行番号を，1000から始まり 100 ずつ増える行番号につけかえます．

以上のような行番号のつけかえに際しては，各行の中のGOTO文・GOSUB文その他に行先として書かれている行番号なども，新しい番号に書きかえられます．

なお，次の点に注意してください．それは，RENUMコマンドを行の順序を変更するのに使うことはできないということです．例えば，10，20，30番という行がある場合，行番号30だけを15とつけかえるという目的で，「RENUM 15，30」とすることはできません．

2·3·5　プログラムを保存する

記憶装置の中にあるプログラムは，コンピュータの電源を切ると消えてしまいます．しかし，電源を切る前にカセットテープにプログラムを記録しておくことができます．そのテープを保存しておけば，また必要なときにテープからコンピュータの中へプログラムを入れて，それをすぐに実行させることができます．これらの方法を以下に説明します．説明事項は次の 3 つです．

① プログラムをテープに記録すること，これを「セイブ」といいます．

② テープからコンピュータの記憶装置へプログラムを入れること，これを「ロード」といいます．

③ テープに記録されているプログラムと，記憶装置内にあるプログラムとを照合検査すること，これを「ベリファイ」といいます．

(1) プログラムのセイブ

次のコマンドにより，記憶装置内のプログラムをカセットテープに記録します.

```
CSAVE    "〈プログラム名〉"
```

プログラム名は6文字以内の長さで，任意の名前を用います．この名前を引用符(")で囲んで書きます．プログラム名を省略することはできません．また，セイブのときにつけた名前は，後にこれをロードするときにも使うので，忘れぬようにメモしておくことが必要です．

カセットテープレコーダにテープをセットし，録音状態にして，CSAVEコマンドを与えます．セイブが終わると，画面にOKが表示されます．

(2) プログラムのロード

次のコマンドにより，テープから記憶装置へプログラムを入力します．

```
CLOAD    "〈プログラム名〉"
```

プログラム名は，セイブのときにつけた名前と同じものを使わなければなりません．

カセットテープレコーダにテープをセットし，再生状態にして，CLOADコマンドを与えます．すると，記憶装置内にそれまで入っていたプログラムは消去されます．そして指定された名前のプログラムがテープ上で見つかると，Found 〈プログラム名〉と画面に表示されます．ロード中は画面の右上部に＊が点滅します．ロードが終わるとOKが表示されます．

(3) プログラムのベリファイ

```
CLOAD ?    "〈プログラム名〉"
```

このコマンドは，テープ上のプログラムと記憶装置内のプログラムとを照合させるものです．プログラムをセイブしたらその直後に，ベリファイを行なって，正しくセイブできたかどうかを確認するとよろしい．

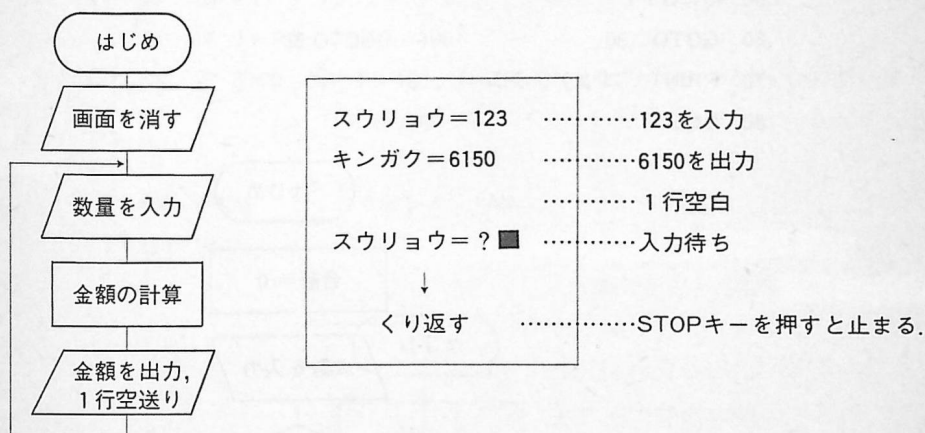
ベリファイするときの操作方法はロードのときと同じです．ベリファイの結果照合に異常がなければ，画面にOKが表示されます．もし異常があった場合はBadが表示されます．

2·4　簡単なプログラム例

以下に掲げるプログラム例は，もっとも基礎的なステートメントを主体として作られています．それらのステートメントの使われ方やプログラムの中で果している役割などを注意して見てください．とくに初心者の方は，これらの例によって，プログラムとはどんなものかを理解してもらいたいと思います．最後に練習問題を掲げてあります．やさしいものばかりですから，挑戦してみてください．

〔例1〕

```
10 REM   キンガク ノ ケイサン      ……………REM文
20 PRINT CHR$ (12)               ……………画面を消す
30 INPUT "スウリョウ="; S
40 K=50*S
50 PRINT "キンガク="; K
60 PRINT                         ……………1行空送り
70 GOTO 30                       ……………30番の行へ行く
```

これは2·1に掲げたプログラム例に修正・追加を施したものです．もっとも大きな相違点は，処理（入力・計算・出力）が何回でもくり返されるということです．また画面の表示の状態も前と違います．画面の状態および流れ図を示しておくので，それを見ながら以下の説明を読んでください．

10番の行はREM文というものです．REM文はコンピュータに動作をさせるための命令文ではなく，単なる注釈文です．人間がプログラムのリストを見たときにわかりやすくするためのメモ書きです．

REMのかわりにアポストロフィ(')を使ってもよろしい．その右には任意の文字を書くことができます．REM文は，プログラムの中のどこにあってもよく，またいくつあってもよろしい，全然なくてもよろしい．

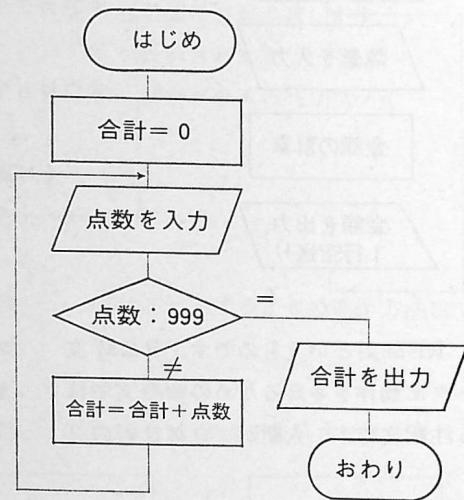
20番の行のようにPRINT文を書くと，画面がきれいに消去されます．そうすればその後の処理が見やすくなります．CHR$というのは文字データを取扱うための関数のひとつです．詳しいことは第6章で説明します．

30番のINPUT文および50番のPRINT文には，引用符(")で囲まれた文字がありますが，これはそのまま画面に表示されます．また60番のPRINT文によって，画面に空白の行（つまり行の間隔）を作ることができます．INPUT文やPRINT文についての詳細は，第3章および第8章で説明します．

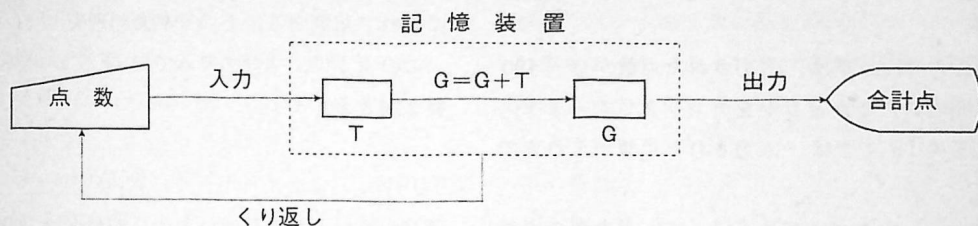
70番の行はGOTO文というものです．これに

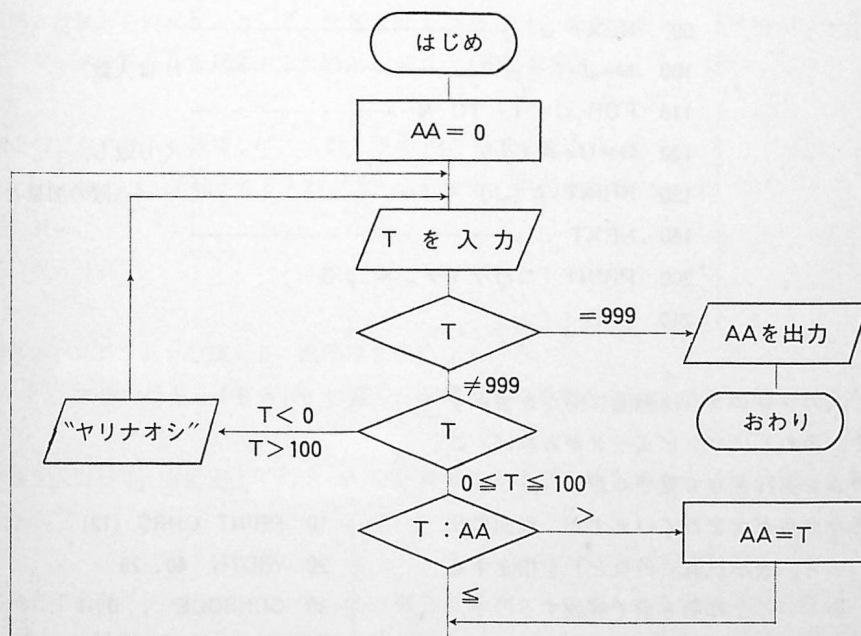
よって，処理の流れは30番の行に移り，また同じ仕事がくり返されることになります．したがって，このプログラムは無限にくり返しを続けることになります．このように処理の流れを変えるための各種の命令文があります．これはプログラムにおいて非常に重要なものです．第4章で詳しく解説します．

〔例2〕 学校のあるクラスでテストが行なわれました．その得点を入力して，合計点を求めることにします．得点は0点から100点の範囲内にあるものとします．また999点はデータの終わりを意味するものとします．

```
10  'ゴウケイテン ヲ モトメル ……REM文
20  G=0
30  INPUT T
40  IF T=999 GOTO 70       ………IF 文
50  G=G+T
60  GOTO 30            …………GOTO 文
70  PRINT "ゴウケイテン=";G
80  END
```

このプログラムも，60番のGOTO文によってくり返しが行なわれます．しかし前の例と違って，特定の条件が満足されたときにくり返しを止めるようになっています．その条件とは，T＝999，すなわち入力された点数が999点に等しいということです．流れ図を見るとわかるように，この条件判断の結果によって流れが分岐します．このような条件判断を行なうのが40番のIF文です．IF文については第4章で解説します．

20番の行のGは合計を求めるために用いられています．手計算でいえばそろばんに相当します．G＝0というのは，そろばんでいえば「ご破算で願いましては」ということです．ただしこのプログラムでは20番の行はなくても差しつかえありません．なぜならば，RUNコマンドによって変数の値はすべてクリアされるからです．

50番のステートメント（代入文）は，そろばん（G）に点数（T）を加算する（たし込む）という仕事をしています．次の図も見ながら，合計を求めるときの考え方を理解してください．

〔例 3〕 前の例では合計点を求めましたが，こんどは最高点を求めてみます．

```
10   REM   サイコウテン ヲ モトメル
20   AA=0
30   INPUT "テンスウ="; T
40   IF T=999  GOTO  100
50   IF T<0  OR T>100  THEN PRINT "ヤリナオシ": GOTO  30
60   IF T>AA THEN AA=T
70   GOTO  30
100  PRINT "サイコウテン="; AA
110  END
```

このプログラムは前の例よりも流れの分岐が多く，やや複雑になっています．そして，前の例では「IF～GOTO」という形を使いましたが，この例では「IF～THEN」が使われています．このよ

うにIF文にもいろいろな形があります.(第4章参照)

50番のIF文では，入力された点数が0～100の範囲内にあるかどうかをチェックしています.

60番のIF文では，入力された点数がそれまでに見つかった最高点よりも大きければ,それをAAに入れて記憶する，という判断処理をしています.

このように，プログラムではIF文が非常に重要な働きをします.

〔例4〕 次のプログラムは，100人未満の点数をすべて記憶してしまい，それから合計点の計算をしています．合計点だけを求めるのならば，すべての点数を記憶しておく必要はありません．しかし，入力した点数を成績順に並べかえるなどの処理をするときは，すべての点数を記憶しておくことが必要になります.

このように，多くのデータを記憶する場合は，「配列」というものを使います．配列とは，複数個の記憶場所をまとめて名前をつけたものです．その個々の場所は番号で呼びます．配列や並べかえなどについては第7章で解説します.

またこのプログラムでは，くり返しをするのにFOR文とNEXT文とを使っています．ここでは説明は加えませんが，第4章で解説するので参照してください.

```
10  REM  ハイレツ ヲ ツカウ
20  DIM  A (100) ---------------------- 配列を設定する.
30  FOR  J＝1  TO  100  STEP 1 ←┐
40  INPUT  A (J)                │ くり返し.
50  IF  A (J)＝999  GOTO  100   │  (点数の入力)
60  NEXT  J ────────────────────┘
100 N＝J−1 ------------------------ N は人数.
110 FOR  J＝1  TO  N ←──────────┐
120 G＝G＋A (J)                 │ くり返し.
130 RRINT  A (J)                │  (点数の加算と出力)
140 NEXT  J ────────────────────┘
200 PRINT "ゴウケイテン＝"; G
210 END
```

〔例5〕 次のプログラムは画面に何かを表示するものです．手もとにコンピュータがあれば，このプログラムを実行させて見てください.

いろいろな命令が含まれていますが，画面の状態（画面の広さ，表示位置，色など）を指定するものです．詳しいことは第8章で解説するので，そちらを参照してください.

```
10  PRINT  CHR$ (12)
20  WIDTH  40, 25
30  CONSOLE  , , 0, 1
40  FOR  K＝1  TO  7
50  LOCATE  K, K : COLOR  K
60  PRINT "COLOR  DISPLAY"
70  NEXT  K
```

以上にいくつかの簡単なプログラム例を掲げましたが，その中で使われているステートメントのうち，もっとも基本的なものをまとめて下に示しておきます．これらは非常によく使われるものですから，初心者はまずこれらを理解してから他のステートメントの学習に進むのがよいと思います．それぞれのステートメントの詳細な解説は，次章以降にあります．

① データの入力（キーボードから）：INPUT文 ………第３章
② データの出力（画面への表示）：PRINT文 …………第８章
③ 演算・代入：代入文 ………第５章
④ 処理の流れの制御：GOTO文，IF文，FOR文，NEXT文，END文 ………第４章
⑤ 配列：DIM文 ………第７章

理解度テスト

次のような処理を行なうプログラムを作りなさい．そして実行させなさい．

(1) 円の半径の長さを入力して，円周の長さと円の面積とを求めて，画面に出力しなさい．（例１参照）

円周＝２×π×半径，面積＝π×半径×半径，π＝円周率＝3.14159

(2) 単価と数量とを何組か入力して，数量合計と金額合計とを求めなさい．単価が０(ゼロ)ならば，データの終わりを意味するものとします．（例２参照）

(3) 例２のプログラムを改造して，人数と平均点とを求めなさい．

〈ヒント〉 次の行を追加すると人数が求められます．平均点はG/Nです．

```
25  N＝0
55  N＝N＋1
```

(4) 例３のプログラムを改造して，最低点を求めなさい．

〈ヒント〉 20番の行を　BB＝100 に直し，60番の行の不等号(＞)の向きを変えたら！

(5) 点数を入力して，合計点・平均点・最高点・最低点および人数を求めるプログラムを作りなさい．

(6) 元金と年利率とを入力して，１年ごとの複利計算で，各年における元利合計を画面に出力しなさい．元利合計が最初の元金の２倍を越えたならば，処理が止まるようにしなさい．

第3章　データの入力

コンピュータの記憶装置の中へデータを入れることを「データの入力」といいます．その方法・形態はいろいろありますが，データを「どこから」入力するかによって次のように分かれます．

① キーボードから手操作によって入力する．
② プログラム中のDATA文に書かれているデータを読む．
③ カセットテープに記録されているデータを入力する。
④ ディスクファイルに記録されているデータを入力する．

本章では上記のうちの①および②について解説します．③，④については第10，11章で説明します．

①のキーボードからの入力には，単純な形から応用形まで各種の方法があります．それらを本章の3·1から3·4までに掲げてあります。プログラムの処理内容に合わせて，入力しやすい形を考えることが大切です．

②については3・5に説明してあります．

3·1　キー入力の基本形

キーボードによるデータ入力においてもっともよく使われる基本的な方法は，INPUT文による方法です．プログラムの中にINPUT文を書いておけば，実行時にデータをキーボードから入力することができます．本節ではこのINPUT文について解説します．第2章（2·1）のプログラム例の中でもINPUT文を簡単に紹介しておいたので，それを復習してから本節の説明を読んでください．

INPUT文の書き方のもっとも単純な形から説明しましょう．

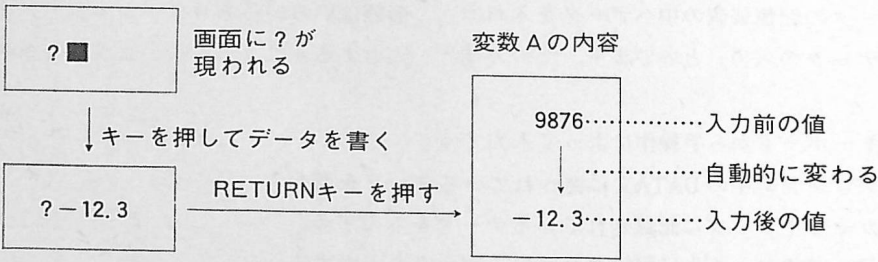
INPUT 〈変数名〉 ………変数名1個だけの形

INPUT文は「キーボードからデータを入力して，指定された変数（記憶場所）へ入れよ」という意味のステートメント（命令文）です．INPUTという語の右に書いてある変数名は，入力したデータを受け入れて記憶しておくべき記憶場所を指定するものです．変数名は数値型でも文字型でもよろしい．

INPUT文が実行されると，その直前まで変数に記憶されていたデータは自動的に消えて，INPUT文で入力されたデータが新らたに入ります．次の例を見てください．

〔例1〕 INPUT A ←── INPUT文（変数名1個だけ）

この例では変数名Aは（末尾に＄記号がついてないから）数値型です．コンピュータがこのINPUT文を実行しようとするときは，画面に疑問符（？）が現われます．これは「データをKey-inしてください」という意味です．そこでキーを押して，？の右側にデータを書き，それからRETURNキーを押せば，そのデータが入力されて変数Aに記憶されます いくらデータを書いても，RETURNキーを押さなければ入力されません．

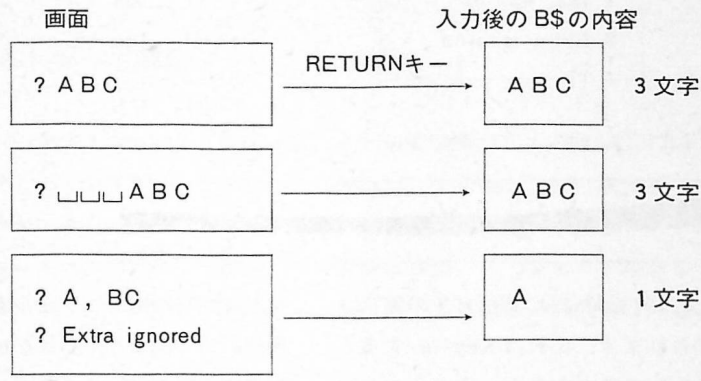
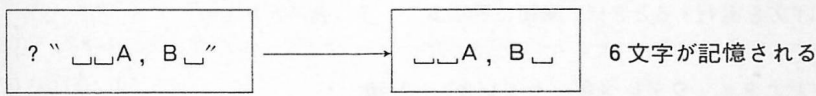
データとして例えば「－12.3」を入力する場合，次図のようになります．

〔例2〕 INPUT B＄ ←── 文字データ1個の入力

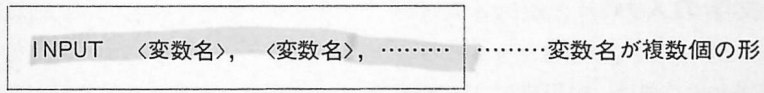
この例では変数名B＄は（末尾に＄記号がついているから）文字型です．画面に？が現われたならば，例えばＡＢＣという3文字を？の右に書いて，RETURNキーを押せば入力されます．このとき変数B＄にはＡＢＣという3文字が記憶されます．

画面に？が現われたとき，何個かの空白を置いてからＡＢＣと書いてRETURNキーを押しても，B＄にはやはりＡＢＣという3文字が記憶されます．すなわち，先頭の空白は無視されて，入力さ

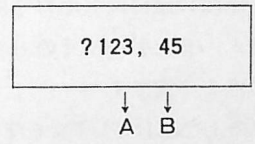
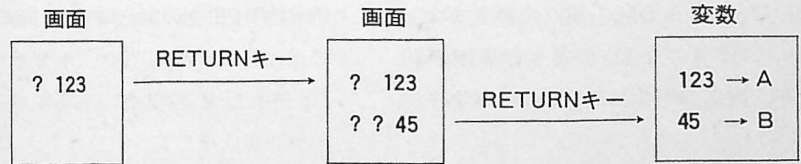
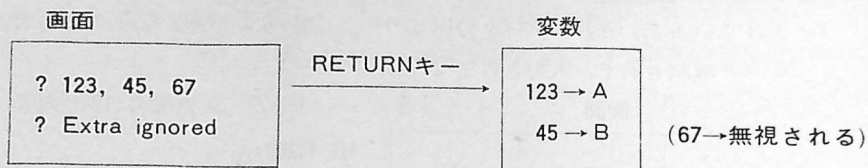
れないのです．

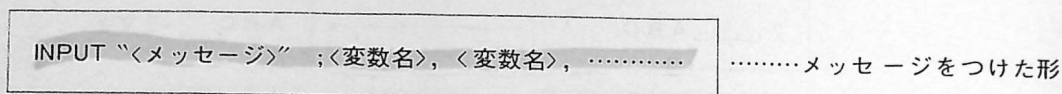
また，入力しようとする文字データの中にコンマ(,)が含まれていると，それはデータの区切りを意味するものと解釈されて，入力されません．このときExtra ignoredというメッセージが表示されます．

これらのことを図示すると次のようになります．

上記のようにコンマや先頭の空白はそのままでは入力できませんが，次図のように引用符(″)で囲めば入力することができます．ただし，引用符そのものは入力できません．

以上は1個のデータの入力でしたが，次のように書けば，ひとつのINPUT文で複数個の変数へデータを入力することができます．任意の個数の変数名をコンマで区切って並べて書きます．

INPUT　〈変数名〉,　〈変数名〉, ………　………変数名が複数個の形

〔例3〕　INPUT A, B

実行時に画面に？が現われたならば，右図のように，AとBとに入力させるべきデータをその順序に，コンマ(,)で区切って書きます．そしてRETURNキーを押せば，変数Aには123，変数Bには45が入力されます．

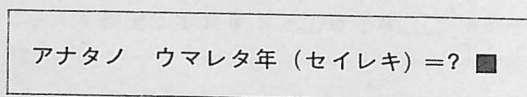
もし下図のように，123（Aの値）だけを書いてRETURNキーを押すと，疑問符が2個(？？)現われます．これは「入力データがまだ不足です！」という意味です．そうしたら45（Bの値）を書いてRETURNキーを押せばよいのです．

また，もし次図のようにデータを多く書き過ぎると，Extra ignoredというメッセージが現われます．このとき，多過ぎたデータ（次図では67）は無視されて，プログラムの実行は継続されます．

さらにINPUT文には，次のような形でメッセージを書くことができます．

INPUT "〈メッセージ〉" ;〈変数名〉，〈変数名〉，………… ………メッセージをつけた形

このINPUT文の実行時には，画面に？の前にメッセージが表示されます．これはKey-in する人に対する案内文として役立ちます．このメッセージのことを「プロンプト文」と呼びます．

INPUT文中にメッセージを書くときは，必ず引用符(")で囲まなくてはなりません．またその右に必ずセミコロン(;)を書かなければいけません．

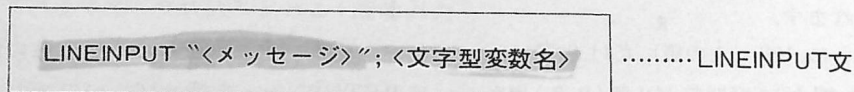
〔例 4〕 INPUT "アナタノ　ウマレタ年（セイレキ）="; Y

このINPUT文を実行するときは，画面に図のように表示されます．

アナタノ　ウマレタ年（セイレキ）=? ■

3·2　特殊文字の入力

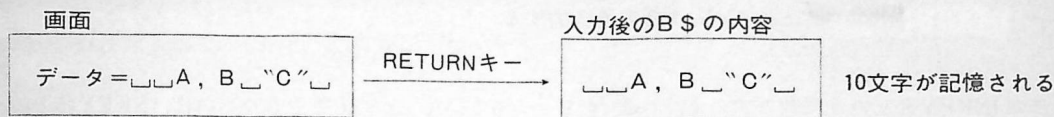
ここで特殊文字というのは，引用符(")・コンマ(,)・空白などのことです．前節で説明したように，INPUT文によって文字データを入力する場合は，コンマや先頭の空白は引用符で囲まなくては入力できません．また引用符そのものは入力できません．この点が不便です．

ところが，次のLINEINPUT文を使えば，これらの特殊文字も普通の文字と同様に，文字データを構成する１文字として入力することができます．すなわち，RETURNキーを押すまでに画面に書いた文字の並びを，区切ることなく，上記の特殊文字も含めて，１個の文字データとして変数に入力します．ただし，入力できる文字データの長さは255文字以内です．

LINEINPUT "〈メッセージ〉";〈文字型変数名〉 ………LINEINPUT文

変数名は，文字型のものを，1個だけ書きます．

メッセージ（プロンプト文）の書き方はINPUT文のときと同じです．省略することもできます．

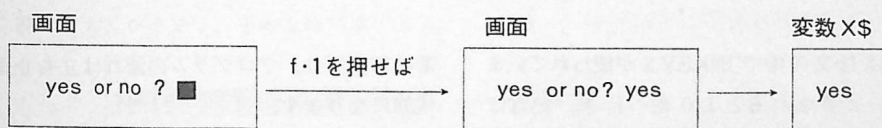
LINEINPUT文の実行時には，画面に疑問符(?)が表示されません．

〔例〕 LINEINPUT "データ="; B$

3·3 ファンクションキーの利用

第1章（1·3）で説明したファンクションキーを，データの入力に利用することができます．KEY定義文によってデータを定義しておけば，データ入力時にファンクションキー1個を押すだけで，15字以内の文字を画面に書くことができます．また，機能文字としてCHR $(13)を含めて定義しておけば，入力時にRETURNキーを押さなくても，ファンクションキーだけでデータを入力できます．このように，入力時のキー操作を速くらくにし，また操作ミスによる誤入力の防止にも役立ちます．

〔例〕

```
10  KEY1, "yes"+CHR$(13) ⎫ KEY定
20  KEY2, "no"+CHR$(13)  ⎭ 義文
     ⁝
100 INPUT "yes or no"; X$…INPUT文
110 IF  X$="yes"  GOTO  1000
120 IF  X$="no"  GOTO  2000
130 GOTO  100
```

100番のINPUT文を実行するときに，画面に？が現われたならば，ファンクションキー1番(f·1)または2番（f·2）を押せば，RETURNキーを押さなくても入力されます．これは，定義されている機能文字CHR $(13)が「PETURNキーを押す」という働きを意味しているからです．

3·4 RETURNキーを使わない入力

キーボードからデータを入力するときは，画面にそのデータを書いて「RETURNキーを押す」のが普通の形です．しかし本節で紹介するINKEY $あるいはINPUT$を使えば，RETURNキーを押さずに入力できます．ただし，入力できるのは文字型データのみであり，INKEY $は1文字，INPUT$は指定した字数だけの文字を入力します．普通の文字だけでなく，機能文字も入力できます．

なお，他の入力方法と違って，画面に疑問符(?)は表示されず，またキーを押してもその文字は画面に表示されません．また，INKEY$とINPUT $はステートメントではなく関数ですから，ステートメントの中に組み込んで用います．

INKEY$ ………1文字を入力する

このINKEY＄という関数は次のように動作します．

「もし，キー（1個）が押されれば，それを文字データとして受け取ってINKEY＄の値とする．もし，キーが何も押されなければ，INKEY＄の値は空（ヌルストリング）となる．」

次の例を見てください．

〔例1〕
```
90  PRINT "Press any key!"
100 X$=INKEY$
110 IF X$="" GOTO 100
120  ～
```
キー1個が押されるまで待つ

90番のPRINT文で画面に「Press any key!」が表示されたとき，何もキーを押さなければX＄の値はヌルストリングになります．110番のIF文では，X＄の値がヌルストリングであるかどうかを比較判断しています．引用符を（間隔をあけずに）2個続けて書いてある（""）のは，ヌルストリングを表わしています．

結局，何もキーを押さずにいれば，100番と110番の行がくり返されて，プログラムの処理は先へ進まずに立ち止まった状態になります．任意のキー1個を押せば，それをINKEY＄が受け取ってX＄に入れます．その結果処理の流れは120番の行へ移り，先へ進みます．

〔例2〕
```
100 IF INKEY$ ="a" GOTO 120
110 GOTO 100
120  ～
```

この例ではIF文の中でINKEY＄が使われています．aのキーが押されると120番へ行き，処理は先へ進みます．a以外のキーを押しても，また100番の行へ戻り，プログラムの流れは立ち止まった状態になります．

次はINPUT＄関数です．

INPUT$(X) ………X個の文字を入力する

カッコの中のXは入力すべき文字数を指定するものです．キーを押してX個の文字を与えれば，それがこの関数の値となります．X個のキーが押されるまでは入力待ちの状態となります．

〔例3〕
```
200 A$=INPUT$ (N)
210  ～
```

N個のキーを押し終わると，そのN文字がA$に代入されます．それから210番の行へ進みます．N個のキーを押し終わるまでは入力待ちの状態であり，先へは進みません．

〔例4〕

```
100 X$=INPUT$(1)
110 IF X$="a" GOTO 1000
120 IF X$="b" GOTO 1500
130 IF X$="c" GOTO 2000
140 GOTO 100
```

1個のキーが押されるまでは，100番の行で待っています．キーが押されると，その文字によって行先が分かれます．

3·5 DATAを読む

前節までの入力方法はいずれもキーボードを用いる方法でしたが，本節の方法は違います．プログラムの中にあらかじめデータを書いておき，実行時にコンピュータがそれを読んで変数に入れるという方法です．ですから「入力」というよりも「割り当て」とでもいった方がよいかも知れません．

仕事の処理のために使われるデータのうちで，あらかじめ値のわかっているものは，プログラムの中に書いておくべきです．そのためには代入文を使うこともできますが，データの個数が多くなると書くのがたいへんで，プログラムも長たらしくなります．このような場合に本節の方法が役立ちます．

プログラムの中にデータを書くためにDATA文を使います．そのデータを変数に入れるためにREAD文を使います．またデータを読み始める位置を指定するためにRESTORE文があります．本節ではこれらを説明します．

DATA 〈定数〉，〈定数〉，………	……DATA文

DATA文には任意の個数の定数をコンマ(,)で区切って並べて書きます．定数は数値型でも文字型でもよろしい．DATA文はプログラムの中のどこに書いてもよく，いくつあってもかまいません．

DATA文に文字定数を書く場合，普通は引用符で囲む必要はありません，しかし，コンマ(,)やコロン(:)あるいは意味のある空白を含む文字定数は引用符(")で囲まなければいけません．

READ 〈変数名〉，〈変数名〉，……	………READ文

READ文には変数名（数値型でも文字型でもよい）を必要な個数だけコンマ(,)で区切って並べて書きます．

READ文とDATA文とは対応して働きます．READ文を実行すると，DATA文に書かれているデータが先頭から順に読まれて，変数に入って記

憶されます．プログラム内にDATA文が複数個あるときは，行番号の若いDATA文から順に読まれます．

ひとつのREAD文を実行し終わったとき,DATA文中のどのデータまで読まれたかをコンピュータは記憶しています．そして次にまたREAD文が実行されるときは，前回のREAD文で最後に読まれたデータのすぐ次のデータから読み始めます.READ文を実行しようとしたとき，読まれるべきデータがなければ，エラー（Out of Data）となります．DATA文中のデータが読まれずに残っても，それは無視されるだけで，エラーにはなりません．

〔例1〕

```
10  DATA  12, 34, 56
20  DATA  ABC, DEF
30  READ  A1, A2 ──────→ A1=12 : A2=34                  } この代入文と同
40  READ  A3, X$, Y$ ──→ A3=56 : X$="ABC": Y$="DEF"    } じ結果になる
50  READ  B ───────────→ Out of Data（エラー）
```

〔例2〕

```
10  DIM  A(4)
20  FOR J=0  TO  4
30  READ  A(J)
40  NEXT  J
50  DATA  100, 80, 70, 60,  0
```

配列A

0	100
1	80
2	70
3	60
4	0

この例では，配列の中へデータをREADしています．このように配列に値を入れるときにもREAD文・DATA文がよく使われます.これについては第7章（7·3）を参照してください．

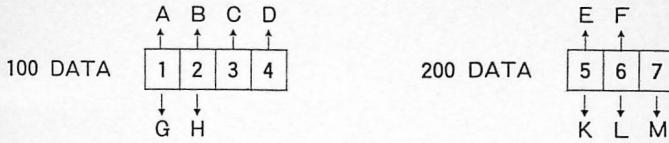
次のRESTORE文は，データを読み始める位置を指定します．これを使えば同じDATA文を2回以上読むことも可能です．

RESTORE 〈行番号〉 ………RESTORE文

RESTORE文が実行されると，その次のREAD文は，指定された行番号のDATA文（の先頭のデータ）から順に読みます．また，RESTORE文の行番号は省略することもできます．その場合は，次のREAD文はもっとも若い行番号を持つDATA文から読み始めます．

〔例3〕

```
10   READ A, B, C, D, E, F
20   RESTORE
30   READ  G, H
40   RESTORE  200
50   READ  K, L, M
100  DATA  1, 2, 3,  4
200  DATA  5, 6, 7
```

これを実行すると，各変数には次のようにデータが対応して入ります．

理解度テスト

(1) 次のINPUT文には，書き方の誤っている所があります．それを指摘しなさい．

① INPUT X;Y;Z

② INPUT キンガクハ；KI

③ INPUT A, B, C, D,

④ INPUT NA$; "ナマエハ"

⑤ INPUT "EIGO NO TENSU=", E

(2) 次の文章のうち，正しい内容のものには○を付け誤りのあるものには×を付けなさい．

① INPUT文の中には，メッセージ（プロンプト文）を必ず書かなければならない．

② ひとつのINPUT文の中に，数値型変数名と文字型変数名とが混在してもよい．

③ ひとつのプログラムの中に，複数個のDATA文を書いてはいけない．

④ READ文を実行するときには，画面に疑問符が表示される．

⑤ RESTORE文を実行するときには，画面に疑問符が2個表示される．

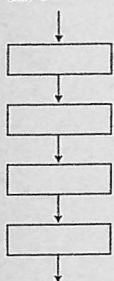
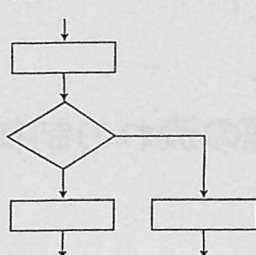
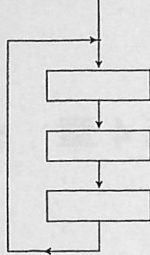
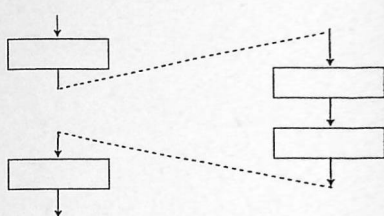
第４章　処理の流れの制御

すでに述べたように，プログラムは行番号の付いたステートメントの集まりです．そして，その行番号の若いものから順に実行されるのが原則です．

しかしこれだけでは，処理の流れは図１のようにきわめて単純な直線的なものであり，複雑な処理はできません．

図２のように，ある条件に対する判断の結果によって，処理の流れを２方向に分けること(分岐)，また図３のように同じ手順をくり返すこと（ループ)，あるいは図４のように，別の所にある処理手順（サブルーチン）へ飛んで，それを実行した後に，またもとの所へ戻ってくる，などのように処

理の流れを制御することが必要になります．

このためのいろいろな命令がBASICには用意されており，それらを組合わせることによって，さらに複雑な処理手順を作ることができます．本章では，これらについて解説します．

4·1　単純な行先指定

プログラム内の各ステートメントは，その行番号の大きさに従って順次に実行されるのが原則です．しかし次に示すGOTO文を使えば，この原則に関係なく無条件に（強制的に)「次に実行すべきステートメント」を指定することができます．

GOTO〈行番号〉

GOTO文に書かれている行番号は「次に実行すべきステートメント」の行番号です．これによって処理の流れは強制的にその行番号のステートメントへ移されます．すなわちこの行番号はGOTOの「行先」です．

「行先」は，そのGOTO文自身より前の方でも後の方でもよろしい．とにかく無条件に他のステートメントを飛越して，指定された行先へ飛んで行きます．そしてその後はそこから，通常の実行順序に従って処理が進められます．そこでこのGOTO文のことを，単純GOTO文とか無条件GOTO文とか無条件飛越し文などとも呼びます．プログラムの中で非常によく使われるステートメントです．

〔例1〕
```
10  INPUT  A  <-----+
      ~             |  無条件飛越し
50  GOTO   10 ------+
60  H=G/N
```

この例の50番のステートメントがGOTO文です．処理の流れは60番へは行かず，10番へ飛びます．

〔例2〕
```
10  INPUT  N
20  GOTO  N  <------------ 誤り
```

行先を指定するための行番号は，必ず数字で書かねばなりません．変数名は使えません．したがってこの例の20番のGOTO文はエラー（Undefined line number）となります．

また，行先がプログラムの中に無い場合も，当然エラーとなります．

4·2　複数個の行先指定

前節の単純GOTO文は，ただ「1個の行先」を指定するだけです．ところが本節で紹介するON～GOTO文は「複数個の行先を用意しておいて，その中のどれかを指定する」というものです．書き方は次のとおりです．

```
ON 〈式〉 GOTO 〈行番号〉,〈行番号〉,…
```

行先を示す行番号は，必要な個数をコンマ（,）で区切って並べます．これを「行先のリスト」と呼びます．(行番号として変数名は使えません.)

ON～GOTO文は次の規則に従って，行先を指定する働きをします．

① まずONの右の式の値を求めます．もし式の値が整数にならない場合は，小数部分を切捨てます．(このようにして求めた式の値をKと呼ぶことにしましょう.)

② 行先リストの中のK番目の行番号を行先として指定し，その行番号のステートメントへ飛びます．(例えばK＝1ならば1番目の行番号を指定し，K＝2ならば2番目，K＝3ならば3番目の行番号を指定します.)

③ K＝0の場合，または「行先リストに並んでいる行番号の個数」よりもKの方が大きい場合は，行先リストは無視されます．そして，ON～GOTO文自身の次にあるステートメントへ進みます．

④ K＜0の場合はエラー（Illegal function call）となります．

〔例1〕
```
10 INPUT"(1, 2, 3)ノドレニシマスカ"; K
20 ON K GOTO 500, 100, 200
30 PRINT "ヤリナオシ"：GOTO 10
```

10番のINPUT文で入力されたKの値に従って，次のように行先が選ばれます．

K＝1ならば行先＝500，K＝2ならば行先＝100，K＝3ならば行先＝200，

K＝4ならば行先＝30，K＝0ならば行先＝30，

K＝－1ならばエラーとなります．

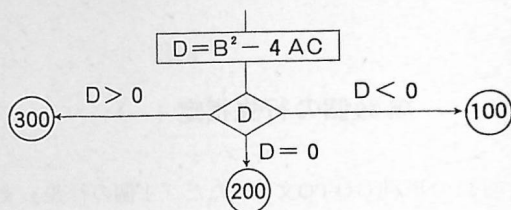
〔例2〕　データの符号（－，0，＋）による行先指定
```
50 D=B*B-4*A*C
60 ON SGN(D)+2 GOTO 100, 200, 300
```

SGN(D)＋2という式の値によって行先を選びます．SGN(D)はDの符号を調べる関数です，第5章(5·4·1)参照．このON～GOTO文は次のように動作します．

Dの値	SGN(D)	SGN(D)＋2	行先
D＜0 ──→	－1 ──→	1 ──→	100
D＝0 ──→	0 ──→	2 ──→	200
D＞0 ──→	1 ──→	3 ──→	300

なお4·3·3〔例1〕を参照してください．

$D=B^2-4AC$

D＞0 → 300　D＜0 → 100　D＝0 → 200

4·3　条件判断による処理

われわれがある行動をするかしないかは，そのときの状況を判断してきめる，ということがよくあります．いいかえれば，状況次第によってなすべき行動が異なるということです．例えば次のようなことです．

もし〈雨が降っている〉{ ならば〈自宅で本を読む〉／そうでなければ〈外出する〉

コンピュータにも，このような行動の仕方をさせることが，しばしば必要になります．そのためにIF文というステートメントが使われます．上のことをIF文式に書くと次のようになります．

IF　雨が降っている　THEN　自宅で本を読むELSE　外出する

このようにIF文は，IF～THEN～ELSE～という形をしています．またELSE～を書かない形も使われます．本節では以下の各項で，次のことを解説します．

4·3·1　IF～THEN型（ELSE以下のない形）
4·3·2　IF～THEN～ELSE型
4·3·3　多重IF型（IF文の中にさらにIF文が含まれている形）

また上例の「雨が降っている」というのは，状況判断の基準となる「条件」です．この条件は「論理式」を使って表わします．論理式を書くための記号や規則がいろいろあるので，それらは次節(4·4)で解説することにします．

4·3·1　IF～THEN型の処理

これはELSE～を使わないIF文の形です．まずこれをよく理解してから，次にELSE～の付いた形（次項4·3·2）へ進んでください．

IF〈論理式〉 THEN〈文〉

……ELSE～のないIF文

論理式は条件を表わすためのもので，式の値は「真」または「偽」となるものです．(詳しくは4·4参照)

このIF文の意味は次のとおりです．

「もし条件が満足されていれば，(いいかえれば，論理式の値が真ならば，) THENの右の文を実行せよ．」

もし条件が満足されていなければ，(すなわち，論理式の値が偽ならば，) THENの右の文は実行しません，

〔例1〕

```
10  IF X>M THEN M=X
20  T=T+X
```

もしX>Mならば，M=Xを実行してから，20番のステートメントへ進みます．

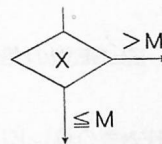
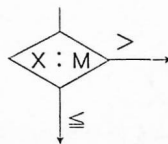
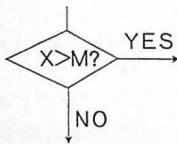
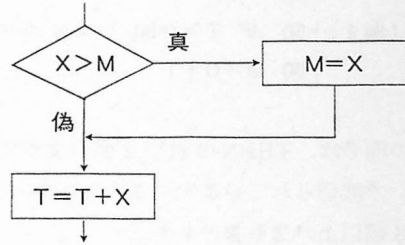
もしX>Mでなければ，M=Xは実行せずに，直接20番のステートメントへ進みます．

この処理を流れ図で表わすと右図のようになります．

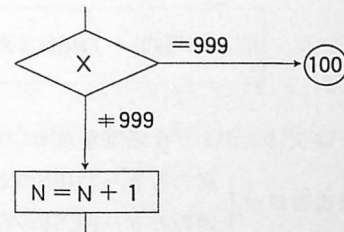
流れ図では条件判断は菱形の記号(◇)で表わします．そこで流れが別方向へ分かれます．これを「分岐」と呼びます．次図のような表現の仕方もできます．

X>M?　YES　NO

X : M　>　≦

X　>M　≦M

〔例2〕

```
20  IF X=999 THEN GOTO 100
30  N=N+1
      ⁝
100 H=G/N
```

X　=999　100　≠999　N=N+1

この例では，THENの右の文としてGOTO文が使われており，〔例1〕とは流れ図の形が違うことに注意してください．

X=999ならば，100番のステートメントへ飛んで行って，行きっぱなしとなります．(30番へは行きません．)

X≠999ならば，100番へは飛ばないで，30番へ進みます．

上例のように，THENの右の文が「GOTO文」の場合は，次のような書き方をしてもよろしい．

① 「GOTO」を省略して，THENの直後に行番号(行先)だけを書く．

IF〈論理式〉 THEN〈行番号〉

……GOTOを省略した形

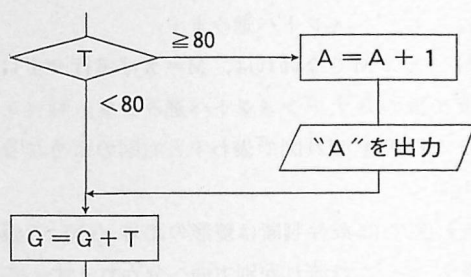
② あるいは，「THEN」を書かずに，論理式の直後にGOTO文を書く．

IF〈論理式〉 GOTO〈行番号〉…これは「IF～GOTO文」とも呼ばれます.

〔例3〕IF X=999 THEN GOTO 100
IF X=999 THEN 100
IF X=999 GOTO 100
} これらは同じ意味になります.

〔例4〕
```
50 IF T>=80 THEN A=A+1 : PRINT"A"
60 G=G+T
```

この例では，THENの右に2個の文がコロン（:）で区切られています．さらにコロンで区切れば3個以上の文も書けます．

この場合の処理は右の流れ図のようになります．すなわち，IFの右の条件が満足されているときは，THENの右の複数個の文が順に実行されます．条件が満足されていなければ，THENの右の文はどれも実行されません．

4·3·2 IF～THEN～ELSE型の処理

これは前項のIF～THEN型の後にELSE～を付け加えたもので，IF文の本格的な形です．

IF〈論理式〉 THEN〈文〉 ELSE〈文〉

このIF文は次のような処理を意味します．

論理式の値が { 真のとき→THENの右の文だけを実行する．(ELSEの右の文は実行しない)
偽のとき→ELSEの右の文だけを実行する．(THENの右の文は実行しない)

「IF～THEN～」までの部分については前項で述べた規則が適用されます．「ELSE～」の部分についてはは次のとおりです．

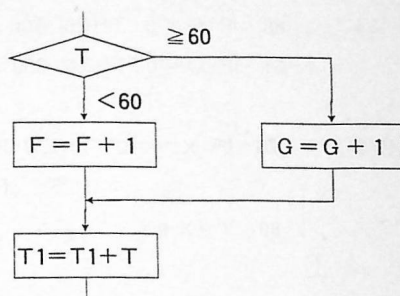
① ELSEの右の文がGOTO文の場合は，「GOTO」を省略して，ELSEの直後に行番号（行先）だけを書いてもよい．しかし，「ELSE」を書かずにGOTO文だけを書く，ということはできない．

………ELSE〈行番号〉………GOTOを省略した形

② コロン（:）で区切って並べた複数個の文がELSEの右にあってもよい．それらの文は，論理式の値が偽のとき順に実行される．

〔例 1〕

```
70  IF  T>=60  THEN  G=G+1  ELSE  F=F+1
80  T1=T1+T
```

右図のように，Tの値が60以上であるか60未満であるかによって，異なった処理を行なった後，80番のステートメントへ合流します.

〔例 2〕 40 IF X<0 THEN GOTO 100 ELSE GOTO 200

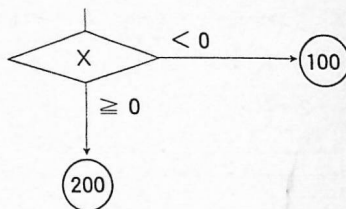
Xの値がマイナスか否かによって，別の行先へ飛びます.

これは次のように書くこともできます.

40 IF X<0 THEN 100 ELSE 200

あるいは，40 IF X<0 GOTO 100 ELSE 200

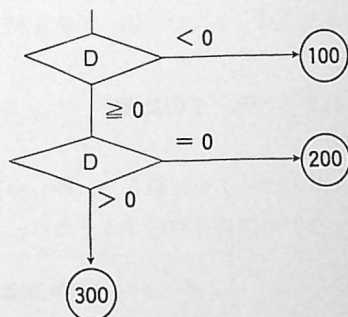
しかし，次の書き方は許されません.(「ELSE」は省略できない)

40 IF X<0 THEN 100 GOTO 200←誤り

4·3·3　多重のIFによる処理

前項までに述べたIF文のTHENの右の文やELSEの右の文として，さらにIF文を書くことが許されています．すなわちIF文の内部にIF文が含まれている形です．これを多重IF文とかIFの重なり（ネスティング）と呼びます.

IFの重なりの深さ（何重になっているか）には制限はありません．したがってかなり複雑な条件判断処理でも，1個のIF文にまとめることができます．しかし無理に重ね過ぎるとややこしくてわかりにくくなり，それがプログラムミスの原因にもなりかねないので，要注意です.

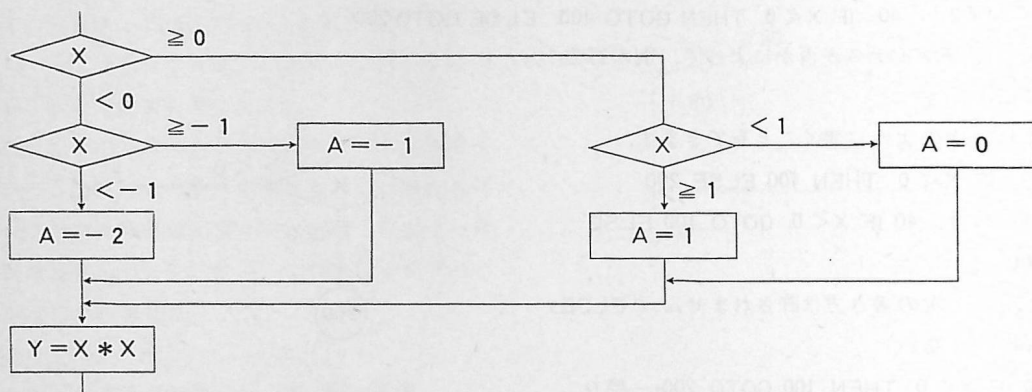
〔例 1〕 これは4·2〔例 2〕と同じ処理を，2 重IF文で表現したものです．比較してみてください.

```
50  D=B*B-4*A*C
60  IF  D<0  THEN  100  ELSE  IF  D=0
    THEN  200  ELSE  300
```

２重IF文を使わなくても，次のように書けば上　と同じ処理になります．

```
50  D=B*B-4*A*C
60  IF D<0 THEN 100
61  IF D=0 THEN 200 ELSE 300
```

〔例２〕

```
70  IF X>=0  THEN IF X<1 THEN A=0 ELSE A=1
               ELSE IF X>=-1 THEN A=-1 ELSE A=-2
80  Y=X*X
```

これを流れ図で表わすと次のようになります．　　IF文と対照して見てください．

IF文が重なるとTHENやELSEの個数が多くなり，どのTHENがどのELSEに対応するのかわかりにくくなってきます．対応の規則は次のとおりですから注意してください．

ELSEは，「それ自身より左にあるTHENのうちで最も近くにあり，かつどのELSEともまだ対応していないTHEN」に対応する．

4·4　論理式

前節で述べたように，IF文において判断のための条件を表わすには「論理式」を用います．論理式とは「真」または「偽」の値を持つ式です．論理式を書き表わすために，関係演算子や論理演算子と呼ばれる記号が使われます．本節ではこれらの記号の種類や書き方などについて解説します．

4·4·1　データの比較

２つのデータを比較して，等しい・大きい・小さいなどの関係を判断するために，次の形の式を使います．

〈データ〉〈関係演算子〉〈データ〉……関係式

これは関係演算子の左側と右側とにデータを書いたものであり,「関係式」と呼ばれます．左右のデータは比較の対象となるものであり，定数や変数名あるいは式を書くことができます．数値データでも文字データでも比較できますが，両者の混合比較（数値型と文字型との比較）はできません．

関係演算子とは比較の記号です．「=，<，>」の3種が基本となりますが，これらの組合わせも使えます．次表のとおりです．

関係演算子	意　味	関係式の例
=	等しい	A=B，A$="XXX"
<	小さい	A<B，A$<B$
>	大きい	A>B，0>A*B
<>	等しくない	A<>B，A−B<>C
><		A><B，A$><B$
<=	小さいかまたは等しい	A<=B，A−B<=C
=<		A=<B，D=<0.001
>=	大きいかまたは等しい	A>=B，SQR(X)>=A
=>		A=>B，A(I)=>A(J)

〔例〕

```
10  A=12:B=45:C=78
20  IF A<B THEN PRINT"TRUE"ELSE PRINT"FALSE"——→TRUEが出力される.
30  IF A=B+C THEN PRINT"シン" ELSE PRINT"ギ"————→ギが出力される.
```

この例では，関係式「A<B」の値は真となります．なぜならば，12<45でありA<Bという関係が成立しているからです．また関係式「A=B+C」の値は偽となります．なぜならば，12≠123であり,「=」の関係が成立していないからです．この「A=B+C」は関係式であり，代入文ではありません．

ひとつの関係式の中に関係演算子と算術演算子（+，−，*，/ など）とが含まれている場合は，算術演算の方が先に実行されます．したがって上の例では，B+Cという加算が先に行なわれて，その値とAとが等しいかどうか比較されるわけです．算術演算子については第5章(5·3) を参照してください．

なお，文字データの比較については6·6を参照してください．

4·4·2　論理演算

前項では関係演算子（比較記号）について述べましたが，本項ではさらに論理演算子(論理記号)を紹介します．関係演算子や論理演算子を含んだ式は総称して「論理式」と呼ばれます．したがって前項の関係式も論理式の中に含まれます．

論理演算子は以下に示すように6種類ありますが，その中でも「AND，OR，NOT」の3種が基本的なものです．まずこれを表にまとめて掲げます．

		X かつ Y	XまたはY	X でない
X	Y	X AND Y	X OR Y	NOT X
真	真	真	真	偽
真	偽	偽	真	
偽	真	偽	真	真
偽	偽	偽	偽	

ここでXとYは，真または偽の値を持つもの，すなわち論理式（関係式を含む）です．

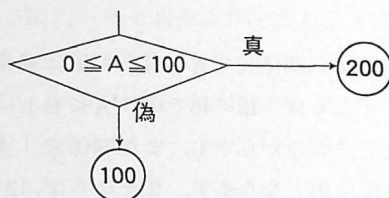
論理演算子「AND」は論理積と呼ばれます．上表を見てわかるように，XとYとがともに真のときに限って「X AND Y」の値は真となります．普通のことばでいえば「Xであり，かつ，Yである」ということです．

論理演算子「OR」は論理和と呼ばれます．XとYのうち少なくとも一方が真のとき，「X OR Y」の値は真となります．すなわち「Xであるか，または，Yである」ということです．

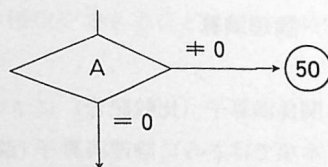
論理演算子「NOT」は否定と呼ばれます．Xの真偽の逆が「NOT X」の値となります．すなわち「Xでない」ということです．

〔例1〕 IF 0=<A AND A<=100 THEN 200 ELSE 100

IFとTHENの間に書かれているのが，AND を使った論理式です．「0=<A」と「A<=100」とは関係式です．この2つの関係式が同時に成立したときに200番へ飛びます．このIF文は次のようにも書けます．

IF A< 0 OR 100<A THEN 100 ELSE 200

〔例2〕 IF NOT A=0 GOTO 50

「Aが0でない」ときに50番へ飛びます．次のようにも書けます．

IF A<> 0 GOTO 50

次に掲げる「XOR, IMP, EQV 」も論理演算子です．この3種は通常あまり必要ありませんが，複雑な関係を調べるとき，あるいは次項で述べるようなビット演算を使うときなどに役立ちます．次表のXやYは上掲のものと同じ意味です．

X	Y	X XOR Y	X IMP Y	X EQV Y
真	真	偽	真	真
真	偽	真	偽	偽
偽	真	真	真	偽
偽	偽	偽	真	真

さて，今までにいろいろな演算子を紹介しましたが，それらを組合わせて長い論理式を作ることができます．その場合に注意すべきことは演算の順序です．ひとつの論理式の中に各種の演算子が使われている場合は，それらを実行する優先順位が次表のように定められています．ただし，同順位のものは，左から右へ向って実行されるのが原則です．また，カッコ(　)でくくられた部分はひとまとまりとして取扱われます．

(1) 算術演算子	① ∧（べき乗） ② －（負数） ③ ＊，／ ④ ￥ ⑤ MOD ⑥ ＋，－
(2) 関係演算子	⑦ ＝，＜，＞
(3) 論理演算子	⑧ NOT ⑨ AND ⑩ OR ⑪ XOR ⑫ IMP ⑬ EQV

〔例3〕

```
10 INPUT A, B, X, Y
20 IF NOT A=B OR X<0 AND Y<0 THEN PRINT"T" ELSE PRINT"F"
```

これを実行して1，2，3，4を入力すると，画面には図のようにTが出力されます．

この長い論理式は次の順序で実行されます．

NOT 1＝2 OR 3＜0 AND 4＜0

1＝2 → 偽，3＜0 → 偽，4＜0 → 偽

NOT 1＝2 → 真，3＜0 AND 4＜0 → 偽

全体 → 真

```
run
? 1, 2, 3, 4
T
Ok
```

論理式があまり複雑になるとわかりにくくなります．次のようにカッコを使うことをおすすめします．

```
20 IF (NOT A=B) OR (X< 0 AND Y< 0) THEN …………
```

4・4・3　ビットの論理演算

前項までの説明では，関係式とか論理式の値は「真」または「偽」である，と述べてきました．それでは，コンピュータの内部では真や偽をどのように表現しているのでしょうか．本節ではこれについて解説します．(この話はビットとか2進法とかが関連するので，初心者の方は読み飛ばして結構です．)

まず関係式の値ですが，コンピュータ内部では「真」を「－1」で表わし,「偽」を「0」で表わします．例えば次のステートメントを実行させてみると，このことがわかります．

```
PRINT  123＝123 ───→ －1が出力される．(真だから)
PRINT  123＞456 ───→   0が出力される．(偽だから)
```

ところで，この－1や0は，実は16ビットの2進法の整数として表現されているのです．すなわち次の形です．(2の補数表示)

```
真→－1→1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
偽→  0→0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

さて次に，論理演算子による演算は次のように行なわれます．

① 論理演算子の左右にある値を，それぞれ16ビットの2進整数に変換します．この値は10進法でいうと－32768～＋32767の範囲に相当します．もしこの範囲を超えていればエラー (Over flow) となります．なおこの変換に際して，小数部分は丸められます．

② 上記の各2進整数 (16ビット) を「1ビットごとに」対応させて，指定された論理演算を行ないます．このとき，ビットが1ならば「真」と解釈し，ビットが0ならば「偽」と解釈します．

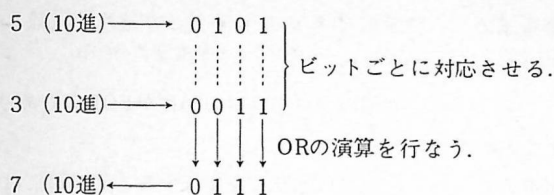
以上の話からわかることは，論理演算子の左右にある値は必ずしも真 (－1) または偽 (0) でなくてもよいということです．下記の例を見てください．

〔例1〕 PRINT　5　AND　3 ───→ 1が出力されます．

```
5 (10進)───→ 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 ┐
                                                ├ビットごとに対応させる．
3 (10進)───→ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 ┘
              ↓ ↓                   ↓ ↓ ↓ ↓ ↓   ANDの演算を行なう．
1 (10進)←─── 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
```

〔例2〕

```
10  A＝5 OR 3
20  PRINT A ───→ 7が出力されます．
```

```
5 (10進)───→ 0 1 0 1 ┐
             ┊ ┊ ┊ ┊ ├ビットごとに対応させる.
3 (10進)───→ 0 0 1 1 ┘
             ↓ ↓ ↓ ↓  ORの演算を行なう.
7 (10進)←─── 0 1 1 1
```

この例では，10番のステートメントは,(5 OR 3) という論理式の値をAに代入するものです．すなわち，Aの右の「＝」は比較記号（関係演算子）ではなく，「代入」の意味になっています．

〔例3〕

```
30 A＝0
40 PRINT A＝5 OR 3 ───→ 3が出力されます.
```

この例では，30番のステートメントの「＝」は代入の意味です．また40番のステートメントのAの右の「＝」は比較記号であり，(A＝5 OR 3)が論理式です．(A＝5) は関係式であり，Aの値は0ですから結局 (0＝5) という比較をすることになります．この比較の結果は「偽」すなわち0となります．したがって (A＝5 OR 3) という論理式は (0 OR 3) ということになり，ビットごとに対応させて，結局3という値になります．

〔例4〕

```
10 IF 5 AND 2 THEN PRINT"T" ELSE PRINT"F"───→F が出力される.
20 IF 5 OR 2 THEN PRINT"T" ELSE PRINT"F"───→T が出力される.
```

論理式 (5 AND 2) の値は0です．すなわち偽です．そこで，Fが出力されます．

論理式 (5 OR 2) の値は7です．このとき，20番のステートメントによって，Tが出力されます．

この例でも見られるように，論理式の値が「0」ならば「偽」と判定され,「0」でなければ (−1でなくても)「真」と判定されます．

4･5 くり返し処理

くり返し処理はプログラムの中で非常によく使います．前節までに説明したGOTO文やIF文などを使えば，くり返しの手順 (ループ) を作りそれを制御できます．しかしその他に，FOR文とNEXT文とを使ってくり返しをする方法があります．これを紹介するのが本節の目的です．

この方法はたいへん便利なものであり，いろいろな場面で利用できます．例えば第7章の配列を使うときなどは，この方法が不可欠だともいえるでしょう．(7･3参照)

以下の解説や例をよく読んで，FOR文とNEXT文を使うときの規則と使い方の要領を理解し，活用してください．

まず次の例を見てください．

〔例1〕 A＝1＋3＋5＋7＋……＋37＋39

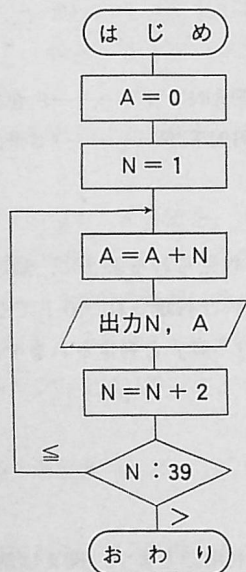
下に掲げたプログラムは，上式のAの値を求めるものです．計算の途中経過も画面に出力するようにしてあります．

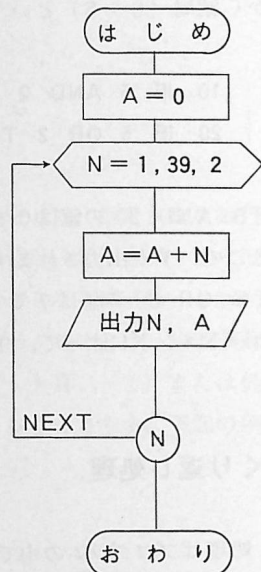
Aは1から39までの奇数の合計です．すなわち，加算を「くり返す」ことによって，Aの値が求められます．このくり返し手順を作るのに，IF文を使う方法と，FOR文〜NEXT文を使う方法とを並べて示してあります．どこが違うか比較してみてください．

右側のプログラムの20番がFOR文であり，50番がNEXT文です．

このプログラムを実行すると，画面には次のように出力されます．

```
run
N = 1      A = 1
N = 3      A = 4
N = 5      A = 9
  ∫          ∫
N =37      A =361
N =39      A =400
Ok
```

―― IF文を使って ――

```
10  A=0
20  N=1
30  A=A+N
40  PRINT "N="; N, "A="; A
50  N=N+2
60  IF N<=39 GOTO 30
70  END
```

―― FOR文を使って ――

```
10  A=0
20  FOR N= 1 TO 39 STEP 2
30  A=A+N
40  PRINT "N="; N, "A="; A
50  NEXT N
70  END
```

FOR文とNEXT文の書き方は次のとおりです．

```
FOR〈変数名〉＝〈始値〉TO〈終値〉STEP〈増分〉
```
……FOR文

```
NEXT〈変数名〉
```
……NEXT文

FOR文とNEXT文とは必ず対(ツイ)にして使います．NEXT文は，対応するFOR文よりも後になければいけません．また，FOR文の変数名とNEXT文の変数名とは同じものを使わねばいけません．この変数のことを「制御変数」と呼びます．

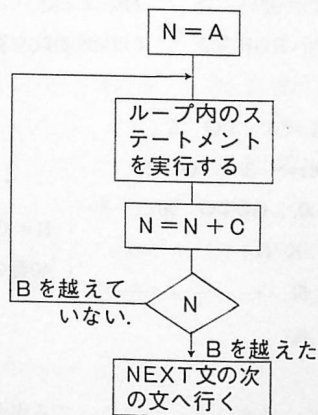
始値，終値，増分には，変数名，定数あるいは式を使うことができます．その値は，正，負，ゼロのいずれでもよろしい．（ただし，増分≠0）

増分が1のときは，FOR文のSTEP以下を省略してもよろしい．すなわち次の形になります．

```
FOR〈変数名〉＝〈始値〉TO〈終値〉
```
←―― STEP以下を省略した形

FOR文とNEXT文は，その間にはさまれたステートメントを，くり返し実行させる働きをします，

```
10  FOR N=A TO B STEP C
      )
50  NEXT N
```

それは，次のような規則に従って行なわれます．

まず，制御変数（N）に始値（A）を代入して，くり返しの第1回目を実行します．それが終わると自動的に，Nの値を増分（C）だけ変化させます．そしてNの値が終値（B）を越えていなければ，くり返しの第2回目を実行します．

以下同様にしてくり返します．

もしNの値が終値（B）を越えたならば，くり返しを止めて，NEXT文の次のステートメントへ移ります．

「終値を越えた」とは，次のことを意味します．

増分＞0の場合は，制御変数＞終値
増分＜0の場合は，制御変数＜終値

〔例2〕

```
10 FOR J=5  TO  1  STEP  1
20 PRINT J;
30 NEXT J
```

この例では増分が－1になっています．すなわち，増分＜0の場合です．くり返しは5回行なわれ，画面には次のように出力されます．

```
5  4  3  2  1
```

〔例3〕

```
10 FOR J=5  TO  1  STEP  1
20 PRINT J;
30 NEXT J
```

この例では増分＞0ですが，終値の方が始値よりも小さくなっています．この場合，くり返しの第1回目だけが行なわれます。

〔例4〕

```
10 FOR N=－3  TO  2
20 PRINT N*N
30 NEXT
```

この例では，NEXT文に制御変数が書いてありません．NEXT文が，それに最も近いFOR文と対応するように処理が流れる場合には，NEXT文の制御変数を省略することができます。

〔例5〕

この例では，ループが2重になっています．3重以上になってもよろしい．このような多重構造を，ループの「ネスティング」といいます．多重になってる場合には，FOR文とNEXT文との対応が適切に行なわれるように注意することが大切です．

4·6　サブルーチンによる処理

ひとつのプログラムの中のあちらこちらで，同じ形の処理をしなければならないことがあります．

これをそのまま書いて行くと，プログラムが長たらしくわかりにくくなり，記憶場所も多く必要になります．このような悩みに対する解決策として，サブルーチンを使うことが有効です．

どのようにするのかというと，上述の「同じ形の処理」をあちらこちらに書かずに，プログラム内の適当な場所に1度だけ書いておくのです．このように書かれたひとまとまりの処理手順がサブルーチンです．そしてそのサブルーチンに書かれている内容を実行させるときは，サブルーチンを呼び出します．呼び出されたサブルーチンはその処理を実行し，実行が終わるともとの場所へ処理の流れを戻します．

ひとつのサブルーチンを，必要に応じて何回でも呼び出すことができます．またひとつのプログラムの中に複数個のサブルーチンを置くこともできます．また，あるサブルーチンがその処理を実行する途中で，さらに別のサブルーチンを呼び出すこともできます．このサブルーチンの重なり（ネスティング）は何重でもよろしい．

サブルーチンを呼び出すにはGOSUB文を使います．またサブルーチンからもとの場所へ処理の流れを戻すにはRETURN文を使います．それらの書き方は次のとおりです．

GOSUB〈行番号〉	……GOSUB文

RETURN	……RETURN文

GOSUB文に書く行番号は，呼び出すべきサブルーチンの先頭のステートメントの行番号です．GOSUB文によって処理の流れはサブルーチンへ飛び，その先頭のステートメントから順に実行されます．

RETURN文は非常に簡単な形をしています．これには行番号は書かないのです．どこの場所へ戻るべきかは，GOSUB文でサブルーチンへ飛ぶときにコンピュータが記憶してくれるのです．したがって，GOSUB文を実行せずにRETURN文に出合うと，コンピュータはエラー（RETURN without GOSUB）を表示します．GOSUB文とRETURN文は対応させて用いなければいけません．ただし，ひとつのGOSUB文に対して複数個のRETURN文が対応してもよろしい．（いいかえれば，ひとつのサブルーチンの中に複数個のRETURN文があってもよいということです.）

〔例1〕

```
100  PRINT"ゴウケイ="; G
110  GOSUB 2000
120  IF W$="1" GOTO 20
  ∫
1990  GOTO 500
2000  PRINT"ツヅケルナラバ1,"                       ┐
2010  RRINT"ヤメルナラバ0ヲオシテクダサイ"           │
2020  INPUT W$                                      ├ サブルーチン
2030  IF W$="1"  OR W$="0"  THEN RETURN             │
2040  GOTO 2000                                     ┘
```

以上に説明したGOSUB文はそれぞれ特定のひとつのサブルーチンを呼び出すものですが，次に紹介するON～GOSUB文は，複数個のサブルーチンの中からひとつを選んで指定することができます．

```
ON〈式〉GOSUB〈行番号〉,〈行番号〉,……
```

これは4·2で解説したON～GOTO文とよく似ており，書き方や動作の規則は同じことです．ON～GOTO文は指定された行先へ飛んで行って行きっぱなしとなりますが，ON～GOSUB文は指定されたサブルーチンへ飛んだ後に，RETURN文によってもとへ戻ることができます．この点が異なるだけです．

〔例2〕
```
50 ON K GOSUB 100, 200, 300
```

このON～GOSUB文は次のものと同様な意味になります．

```
50 IF K= 1  THEN GOSUB 100
51 IF K= 2  THEN GOSUB 200
52 IF K= 3  THEN GOSUB 300
```

4·7 実行の停止と再開

プログラムの中に次のEND文またはSTOP文を書いておくと，そこでプログラムの実行は止まります．

END文やSTOP文はプログラムの中のどこにあってもよく，またいくつあってもかまいません．また，プログラム中にEND文やSTOP文がなくても，実行すべきステートメントが無くなれば，自動的に止まります．

```
END   ……………END文
```

END文を実行するとプログラムの実行は止まり，画面に「Ok」が表示されます．そしてコンピュータはコマンドレベルに戻ります．プログラムの末尾に必ずしもEND文はなくてもよろしい．

```
STOP  ……………STOP文
```

STOP文を実行するとプログラムの実行は止まり，画面に「Break in〈行番号〉」が表示され，またブザーが鳴ります．そしてコマンドレベルに戻ります．表示される行番号はSTOP文の行番号ですから，プログラム中に複数個のSTOP文があっても，どこで止まったかがわかります．

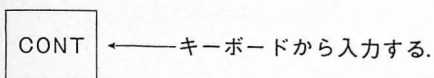
END文やSTOP文とは関係なく強制的にプログラムの実行を止めるには，キーボードのSTOPキ

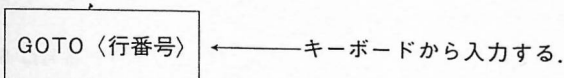
ーを押します．なお，INPUT文の実行時(データを入力しようとしているとき）には，CTRLキー（コントロールキー）とCのキーとを同時に押しても止まります．いずれの場合も「Break in〈行番号〉」が表示され，コマンドレベルに戻ります．

また，ESCキー（エスケープキー）を押してもプログラムの実行が止まります．(ただしINPUT文実行時には止まらない.）この場合は画面にカーソルだけが表示され，入力待ちの状態になります．そのとき，「STOPキーまたは(CTRL＋C)キー」以外のキーを押せば，プログラムの実行が再開され継続して処理が行なわれます．

上述のようにEND文やSTOP文あるいは各種のキーによってプログラムの実行が止まった時点では，記憶装置内の各変数の値は消えずに残っています．そこで，直接モードの命令によって変数の値を画面に表示して確認したり，変数の値を変更したりすることができます．

ESCキーで実行を止めたときにまた再開できることは上で述べましたが，それ以外の仕方で実行が止まった場合でも，次に示すCONTまたはGOTOという命令（直接モード）によってプログラムの実行を再開することができます．

CONT	←―キーボードから入力する.

画面にCONTと書いてRETURNキーを押せば，プログラムの実行が再開されて，継続して処理が行なわれます．なお，この命令を入力するにはファンクションキー10番（SHIFT＋f5キー）を利用できます．

GOTO〈行番号〉	←―キーボードから入力する.

これを画面に書いてRETURNキーを押せば，指定した行番号のステートメントからプログラムの実行が再開されます．

以上のようなプログラムの実行停止や再開は，プログラムのデバッグのときなどに役立ちます．

ただし，実行を止めてから再開するまでの間に，プログラムの内容（ステートメント）を変更することはできません．もし変更してから再開しようとすると，エラー（Can't continue）となります．

〔例〕

```
10 A=12:B=45
20 PRINT A;B
30 STOP
40 PRINT"STOP-CONT"
50 END
60 PRINT"END-CONT"
```

このプログラムを実行して止まったとき，CONT命令によって再開すると，画面は図のようになります．

```
run
  12  45
Break in 30
Ok
cont
STOP－CONT
Ok
cont
END－CONT
Ok
```

4·8 エラー処理

プログラムの実行中にエラーが発生すると，通常はエラーメッセージを画面に表示して，プログラムの実行を止めてコマンドレベルに戻ってしまいます．これは，場合によってはたいへん不便なことです．

本節でこれから解説する方法を用いると，エラーが発生した場合にすぐに実行を停止せずに，何らかの処理をしたうえでプログラムの実行を継続させることができます．

例えば次のプログラムを見てください．

```
  ∫
100  INPUT "A="; A%
110  INPUT "B="; B
120  Y=SQR (A%)
130  PRINT  A%; B
  ∫
```

変数A％は整数型ですから，32767より大きい値を入れることはできません（第5章参照)。ところが，誤って32799というような値を入力すると，次のようなエラーメッセージが出て，実行が止まります．

Overflow ………エラー番号は6．

また，A％の値がマイナスの場合は，120番の代入文を実行するときに，次のようなエラーメッセージが出て，実行が止まってしまいます。(SQRは平方根を求める関数です．第5章参照)

Illegal function call ………エラー番号は5．

エラーにはこの他にもいろいろなものがあり，それぞれにエラー番号が付いています．付録のエラーメッセージ表を参照してください．

さて，次の例は，上のプログラムにいくつかの命令を追加して，エラー処理を行なうようにしたものです．

〔例1〕

```
10   ON ERROR GOTO 1000
      ∫
100  INPUT "A=" ; A%
110  INPUT "B=" ; B
120  Y=SQR (A%)
130  PRINT A%; B
      ∫
1000 IF ERR=6 AND ERL=100 THEN PRINT "INPUT AGAIN !":RESUME
1010 IF ERR=5 AND ERL=120 THEN PRINT"SQR(マイナス)!":RESUME NEXT
1020 ON ERROR GOTO 0
```

このプログラムを実行して，A%に32799を入力すると，画面は次のようになります．

```
A= ?  32799        ←――誤って32799を入力する
INPUT  AGAIN !
A= ?  327          ←――327を入力しなおす
B= ?  5
 327   5
```

また，このプログラムを実行して，A%に－2を入力すると，INPUT文ではエラーは発生しませんが，120番の代入文でエラーが起ります．実行すると画面は次のようになります．

```
A= ?  -2
B= ?  5
SQR (マイナス) !
-2     5
```

この例のように，エラー処理の手順をプログラムに組込んでおくと，エラーが発生しても，実行を止めずに続けることができます．このために用いた命令について，以下に説明を加えます．

上例の1000番以降の行がエラーの処理の仕方を指示する手順であり，エラー処理ルーチンと呼ばれます．エラーが発生したときにエラー処理ルーチンへ行かせるために，次のステートメントを用います．

ON ERROR GOTO 〈行番号〉

この行番号はエラー処理ルーチンの先頭の行の番号（上例では1000)です．ON ERROR GOTO

文を実行すると，コンピュータはこの行番号を記憶します．記憶されたこの行番号は，プログラムの実行が正常に終了した後でも，まだ消えずに記憶され続けます．この記憶を消すには，次のように行番号を0に指定したON ERROR GOTO 文を実行すればよろしい．

```
ON ERROR GOTO 0
```

上例のプログラムでは，1000番および1010番の行では2種類のエラーのみを調べて処理しています．もしこれ以外のエラーが発生した場合は，1020番の行が実行されます．すると，コンピュータは，該当するエラーメッセージを表示して，プログラムの実行を停止しコマンドレベルに戻ります．

なお，エラー処理ルーチンを実行中にさらにエラーが起きた場合は，それに対するエラーメッセージが表示されて，プログラムの実行は停止します．

次にERRおよびERLについて説明します．これらはBASICシステムが持っている特別な変数です．

ERRは，エラーが発生するとそのエラー番号を記憶します．またERLは，エラーが発生したステートメントの行番号を記憶します．これらは，上の例のように，エラー処理ルーチンのIF文の中で利用されます．

エラー処理ルーチンから抜け出して，また正常な処理の流れに戻るために，次のRESUME文を用います．これには3つの形があります．

①
```
RESUME
```

このRESUME文を実行すると，エラーの原因となったステートメントから，プログラムの実行が継続されます．

②
```
RESUME NEXT
```

このRESUME文を実行すると，エラーの原因となったステートメントのすぐ次のステートメントから，プログラムの実行が継続されます．

③
```
RESUME 〈行番号〉
```

このRESUME文を実行すると，指定された行番号の所から，プログラムの実行が継続されます．

上の例で取扱ったエラー番号は，BASICシステムにおいてあらかじめ設定されている番号でした．しかし次の例のようにすると，使用者がエラー番号をプログラムの中で定義することができます．

〔例2〕

```
10 ON ERROR GOTO 1000
        ⌇
100 INPUT "A="; A
110 IF A<0 THEN ERROR 250
        ⌇
1000 IF ERR=250 AND ERL=110 THEN PRINT"Try again!":RESUME 100
1100 ON ERROR GOTO 0
```

この例では，110番のIF文の中でエラー番号を定義しています．すなわち，A＞0ならば，それは250番というエラー番号を持つエラーが発生したことになり，エラー処理ルーチン（1000番）へ飛びます．

1000番の行のRESUME文では，戻り先として行番号100を指定しているので，また100番のINPUT文が実行されます．すなわち，上述のRESUME文の③の形です．

ERROR〈エラー番号〉

……ERROR文

ERROR文によって使用者が定義するエラー番号は，255以下の番号とし，またBASICシステムにおいてすでに設定されているエラー番号よりも大きなものを用います．上例では250番を用いています．

次図は，上例のプログラムを実行して，Aに－123を入力したときの画面の状態を示しています．

```
A=? -123
Try again!
A=?
```

理解度テスト

(1) 次のステートメントの書き方は，文法的に正しいか．正しければ○をつけ，誤りならば×をつけなさい．

① IF A＜0 THEN 20 ELSE 100

② IF A＞0 N=N+1 AND GOTO 200

③ IF A＜0 THEN A=0

④ IF A=B+C GOTO 200 ELSE 300

⑤ IF (A=0, B=0) GOTO 100

⑥ IF DATA="END" GOTO 999

⑦ IF X=－1 THEN STOP

⑧ IF X$="END" THEN END

⑨ IF A=1 OR A=2 GOTO 100 OR 200

⑩ IF K>1 THEN GOSUB 1000

⑪ GOTO NEXT

⑫ ON A=1, 2, 3 GOTO 10, 20, 30

⑬ ON GOSUB 500

⑭ FOR J=1, 49, 2

⑮ FOR A=49 TO 1

(2) ある学校では，テストの点数が60点以上ならば合格とし，60点未満ならば不合格としています．1人分の点数を入力して，もし合格ならば「ゴウカク」,もし不合格ならば「ヤリナオシ」と画面に表示するプログラムを作りなさい．

画面への表示は，次のようなPRINT 文を用いればできます．

PRINT "ゴウカク" あるいはPRINT "ヤリナオシ"

(3) ある学校では，テストの点数によって，成績を下記のように4ランクに分けています．1人分の点数を入力して，どのランクになるかを調べ，そのランク（A，B，C，D）を画面に表示するプログラムを作りなさい．

A：100点～80点，B：79点～70点，C：69点～60点，D：59点～0点

(4) うるう年の定め方は次のようになっています.「西暦年数が4で割りきれる年はうるう年とする．ただし，100で割りきれる年のうち，(年数÷100)がさらに4で割りきれない年は平年とする.」

さて，西暦年数を入力して，うるう年か平年かを調べるプログラムを作りなさい．

割りきれるかどうかを調べるためには,MODという演算を利用すると便利です．MODについては，第5章（5·3）を参照してください．

(5) 次の級数の和を求めるプログラムを作りなさい．

① $2^2+4^2+6^2+\cdots\cdots+38^2+40^2$

② $1+2^1+2^2+2^3+\cdots\cdots+2^{12}+2^{13}$

③ $1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\frac{1}{5}-\cdots\cdots+\frac{1}{99}-\frac{1}{100}$

④ $\frac{1}{1\times2}+\frac{1}{2\times3}+\frac{1}{3\times4}+\cdots\cdots+\frac{1}{50\times51}$

⑤ $1+\frac{1}{1}+\frac{1}{1\times2}+\frac{1}{1\times2\times3}+\frac{1}{1\times2\times3\times4}+\cdots\cdots+\frac{1}{1\times2\times3\times4\times5\times6\times7\times8\times9}$

第5章　数値データの演算

数値データの演算はプログラムの中で頻繁に行なわれる重要なものです．コンピュータに演算を指示するために式を用います．式は変数名，定数，関数，あるいは演算記号によって構成されます．また式の演算によって得られた値を変数に記憶させるために代入文が必要です．

これらのものについて本章で解説します．

5・1　数値データの型

数値データには次の3つの型があります．

① 整数型
② 単精度実数型（略して，単精度型と呼ぶ）
③ 倍精度実数型（略して，倍精度型と呼ぶ）

②と③をまとめて実数型と呼ぶ．

これらの3つの型は，コンピュータ内部での記憶形式が異なり，それにともなって数値の精度や

取扱い方あるいはプログラムの書き方が違ってきます．これは変数に対しても定数に対してもいえることです．

そこでまず，この3つの型の区別を表現する方法を知らなくてはなりません．本節では，5·1·1で変数の型を指定する方法を説明し，5·1·2で各型の定数の書き方を説明します．

5·1·1　変数の型の宣言

変数の型の宣言とは，ある変数が何型であるかを指定することです．指定の仕方は次のようになっています．

変数の型を
- 宣言するには
  - 方法①：型宣言文字を変数名の後に付ける．
  - 方法②：型宣言文を書いておく．
- 何も宣言しないと──→単精度型の数値変数とみなされる．（暗黙の型宣言と呼ばれる．）

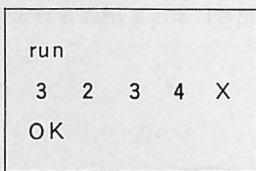
普通のプログラムでは単精度型を使うことが多いので，その分に対してはとくに宣言をしなくても間に合うわけです．しかし整数型や倍精度型を使おうとするときは，必ず方法①または方法②によって積極的に宣言しなければなりません．

方法①は，型宣言文字と呼ばれる特定の文字を，変数名の後に付けて型を指定する方法です．型宣言文字には次の4種があります．（文字型の分も併記しておきます．）

型宣言文字	指定される型	例
%	整数型	A%，A1%
!	単精度型	A!，AA!
#	倍精度型	A#，AX#
$	文字型	A$, B2$

〔例1〕
```
10  A=1 :A%=2:A!=3:A#=4:A$="X"
20  PRINT  A; A%; A!; A#; A$
```

これを実行すると，図のように画面に出力されます．これを見ると，変数AとA!とは同じものを指定しており，その他の変数はすべて異ったものであることがわかります．

```
run
 3   2   3   4   X
OK
```

型宣言の方法②は次の型宣言文を用いるものです．

```
DEF <型の指定> <文字範囲の指定>
```

ここで型の指定というのは，次表に掲げた4種です．これをDEFに続けて（空白を置かずに）書きます．

型の指定	指定される型
INT	整数型
SNG	単精度型
DBL	倍精度型
STR	文字型

文字範囲の指定は英文字で書きます．次の例を見てください．

〔例2〕 10 DEFINT A

これは文字範囲の指定がAとなっています．したがって最初の文字がAの変数名（A, AB, A1など）は整数型（型の指定がINTだから）と宣言されたことになります．

〔例3〕 10 DEFDBL M−P ←—アルファベットのMからPまでの範囲を指定
　　　 20 N%＝123

この型宣言文では，最初の文字がM，N，O，Pのいずれかである変数名は，すべて倍精度型となります．

ただし，20番のステートメントのN%は，頭文字がNであっても整数型となります．すなわち，型宣言文字による型指定は型宣言文による指定に優先するのです．

変数の型を宣言する方法は以上のとおりですが，次に，各型の変数の記憶形式や精度などについて説明を加えておきます．

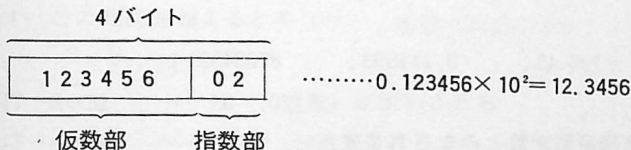
(1) 整数型

1個の整数型データの値は，コンピュータ内部では2バイト（16ビット）で表現されます．すなわち，16桁の2進数の形で変数に記憶されたり演算されたりします．ただし，マイナスの値は補数表示となり，頭の1ビットは符号桁の役割をします．

整数型データの値は（−32768〜32767）の範囲内に限られます．この範囲を超えた値を整数型として取扱おうとするとエラー（Over flow）となります．

2^{15}＝32768なので，このような制限範囲になるわけです．

(2) 単精度型

1個の単精度型データの値は，コンピュータ内部では4バイト（32ビット）で表現されます．その表現形式は次のようになっています．例えば12.3456は，

というように，仮数部と指数部とに分けて表わされます．

ただし，上記は表現の考え方を説明するためのものであり，実際には，これを2進数（を若干変形）の形で表現しています．プログラムの中に10進法で書いてある定数，あるいはINPUT文で入力されたデータなどは，このような2進法による表現に自動的に変換されて，コンピュータ内部で

記憶されたり演算されたりします．

一般に10進数を2進数に変換する際には，変換誤差が生ずる可能性があります．例えば0.1（10進数）を2進数に変換すると，0.000110011001100…という無限小数になります．これを有限の桁数で打ち切って記憶するので，変換誤差が生じます．

したがって単精度型（倍精度型も同様）のデータは，プログラムに書かれた値あるいは入力した値と，コンピュータ内部で記憶された値との間に若干の差異があり得る，ということを承知しておかねばなりません．単精度型の仮数部は，10進法に換算して約7桁の精度で記憶されます．

また単精度型データの絶対値は，1.70141×10^{38} を超えてはいけません．これを超えると，エラー（Over flow）となります．

(3)　倍精度型

1個の倍精度型データの値は，コンピュータ内部では8バイト（64ビット）で表現されます．その表現形式は単精度型の場合と同じ考え方ですが，仮数部の精度は10進法に換算して約16桁となります．したがって高い精度で計算したい場合には，倍精度型を使うべきです．

5·1·2　数値定数の書き方

プログラムの中に数値定数を書くとき，あるいはINPUT文の実行時に画面にデータを書くときには，以下に述べるような形式で定数を表現することができます．

(1)　基本型定数

これは日常われわれが用いている数値の書き方と同じです．数字を並べて書き，必要ならば小数点(．)あるいは符号(－，＋)を併用します．コンマ(，)を含むことは許されません．

符号は左端に付けます．プラス符号(＋)は省略することができます．いいかえれば，符号が付いていなければ正の値とみなされます．

また，末尾に型宣言文字を付けることができます．型宣言文字とは前項（5·1·1）で述べたものです．型宣言文字が付いていると，それによって型が指定されます．

末尾に型宣言文字が付いていないときは，書かれている有効数字の桁数によって，次のように解釈されます．

有効桁が7桁以下のとき ──→ 単精度型 ⎫
有効桁が8桁以上のとき ──→ 倍精度型 ⎭ のデータとみなす．

〔例1〕　次のものは単精度型定数とみなされます．

1234567，　－123.45，　3.141593，　87654321！

〔例2〕　次のものは倍精度型定数とみなされます．

12345678，　－123.45＃，　3.14159265358979，　87654321＃

〔例3〕　次のものはコンマを含んでいるので誤りです．

12，345，678 ………誤り

(2) 指数型定数

指数型というのはEまたはDという文字を使った表現です．EやDは大文字で書きます．Eならば単精度型，Dならば倍精度型とみなされます．

EやDの左側の部分を仮数部と呼び，右側の部分を指数部と呼びます．指数部は（−38〜38）の範囲の整数で書きます．次の例を見てください．

〔例 4〕 12.345E＋2 ⟶ 12.345×10^2＝1234.5の意味
12.345E2 ⟶ 指数部の正符号(＋)は省略してもよい
12.345E−2 ⟶ 12.345×10^{-2}＝0.12345 の意味
−12E3 ⟶ −12×10^3＝−12000
−12D−3 ⟶ −12×10^{-3}＝−0.012 ……Dだから倍精度型
1.70141E39 ⟶ 指数部が大き過ぎる……エラー (Over flow)

(3) 16進型定数

16進型定数とは，&Hという文字の後に16進数を書いたものです．この16進数は0〜FFFFの範囲の整数値とします．この範囲を超えるとエラー(Over flow）となります．また，7FFFを超えた場合は，補数表示とみなされて，マイナスの10進数に対応します．

〔例 5〕 PRINT &H 32 ⟶ 50（10進数）が出力される
PRINT &H 7FFF ⟶ 32767（10進数）
PRINT &H 8000 ⟶ −32768（10進数）
PRINT &H 8001 ⟶ −32767（10進数）
PRINT &H FFFF ⟶ −1（10進数）

(4) 8進型定数

8進型定数とは，&または&Oの後に8進数を書いたものです．この8進数は0〜177777の範囲の整数値とします．この範囲を超えるとエラー(Over flow）となります．また，77777を超えた場合は，補数表示とみなされて，マイナスの10進数に対応します．

〔例 6〕 PRINT &O 32 ⟶ 26（10進数）が出力される
PRINT &77777 ⟶ 32767（10進数）
PRINT &100000 ⟶ −32768（10進数）
PRINT &177777 ⟶ −1（10進数）
PRINT &H32＋&32＋32 ⟶ 108（10進数）

5·2　代入文

代入文はプログラムの中で非常によく使われるステートメントです．次の形に書きます．

```
LET 〈変数名〉＝〈式〉
```

代入文の中の LETという語は省略してもよろしい．すなわち次の形になります．

```
〈変数名〉＝〈式〉
```

〔例1〕 LET　A＝B＋C　…………………………　｝どちらも同じ意味です．
　　　　A＝B＋C　……LETを省略した形

第2章（2·1）でも述べたように，代入文の変数名と式との間にある「＝」は，左辺と右辺とが等しいという意味ではなく，「代入」ということを表わすものです．すなわち代入文は，

「右辺の式の値を，左辺の変数（記憶場所）に入れて記憶せよ」

という意味のステートメントです．

代入文を実行すると，その直前まで左辺の変数に記憶されていたデータは自動的に消えて，新たに右辺の式の値が入ります．このとき，左辺の変数以外の変数の値は変化しません．

〔例2〕

```
10  X=123      ──→ 変数Xに123という値を代入する
20  PRINT  X   ──→ 123が出力される
30  A=45       ──→ 変数Aに45という値を代入する
40  B=A        ──→ 変数Bに，Aの値（45）を代入する
50  X=A+B      ──→ 45+45，すなわち90をXに代入する
60  PRINT  X   ──→ 90が出力される
```

10番の代入文でXの値が123になりますが，50番の代入文を実行すると，Xにあった123という値は自動的に消えて，新たに90が入ります．

代入文の左辺には変数名を1個だけ書きます．複数個は書けません．

また右辺の式は，変数名，定数，関数および演算記号で構成します．ただし，演算記号が全然含まれていなくても（変数名1個だけ，あるいは定数1個だけでも），それを「式」とみなします．

代入文の中の変数名や式は，数値型でも文字型でもよろしい．ただし，右辺と左辺の型は一致していなければなりません．すなわち，数値型代入文（算術代入文という）と文字型代入文とがあるわけです．文字型データの演算については第6章で解説します．

〔例 3〕 次の代入文は，左辺に変数名が複数個あるので誤りです．

A，B＝C＋D ──→ エラー（Syntax error）

〔例 4〕 次のものは数値型代入文（算術代入文）です．

Y＝A＊X＋B ……………………右辺の式は $ax+b$ の意味
C＝SQR(A＊A＋B＊B) ………右辺の式は $\sqrt{a^2+b^2}$ の意味
G＝0 ……………………Gの値を0(ゼロ)にする
N＝N＋1……………………Nの値を1だけ増加させる

〔例 5〕 次のものは文字型代入文です．

A$＝B$
A$＝B$＋C$
X$＝"TOKYO"

〔例 6〕 次のものは，左辺と右辺の型（数値型か文字型か）が一致していないので誤りです．

X＝"TOKYO"
A$＝B＋C
} エラー（Type mismatch）

数値型代入文の左辺と右辺の型（整数型，単精度型，倍精度型）が異なっている場合は，右辺の式の値は，左辺の変数の型に自動的に変換されて代入されます．左辺の変数が整数型の場合は，代入される値が（−32768〜32767）の範囲にないと，エラーになります．

〔例 7〕

```
10  A%＝45.6：PRINT A% ──→ 45が出力される
20  B%＝−45.6：PRINT B% ──→ −46が出力される………要注意
```

左辺の変数が整数型の場合，値の丸め方に注意してください．CINT関数（5·4·1）を参照のこと．

〔例 8〕

```
10  A#＝87654321！
20  B#＝87654321#
30  PRINT A# ──→ 87654320が出力される
40  PRINT B# ──→ 87654321が出力される
50  PRINT A#−B# ──→ −1が出力される
```

10番の代入文の右辺の定数は！が付いているので単精度型です．したがってコンピュータ内部では，7桁分の精度でしか扱われません．左辺の変数が倍精度型であっても，そこに代入されるデー

タが単精度型であれば，7桁分の精度しか得られません．

なお，このプログラムを画面にLISTすると，10番のステートメントは次のように表示されます．

10　A#=8.76543E+07

5·3　式による演算

式は，変数名，定数，関数，あるいは，それらと演算記号（演算子ともいう）との組合わせで構成されます．式は代入文の右辺だけでなく，PRINT文その他のステートメントの中でも使われます．式には次のような種類があります．

① 算術式（数値式）

② 文字式

③ 論理式（関係式を含む）

本節では①について解説します．②は第6章を，③は第4章を参照してください．

算術式は数値型の値を持つ式です．算術式で用いる演算記号（算術演算子）を次表に示します．

算術演算子	意　　味	式の例
＋	加 算	A＋B
－	減算,マイナス符号	A－B，－A
＊	乗 算	A＊B
／	除 算	A／B
∧	べキ乗	A∧B
¥	除算(整数扱い)	A¥B
MOD	剰余(整数扱い)	A MOD B

算術演算子には演算実行の優先順位が次表のように定まっています．同順位のときは式の左から右へ向って実行するのが原則です．

なお，式の中に丸いカッコ（　）を用いることができます．カッコは重ねてもよろしい．カッコの中はひとまとまりのものとして扱われます．式の中の演算順序がややこしい場合には，積極的にカッコを用いるとわかり易くなります．

優先順位	演　算　子
1	∧
2	－(マイナス符号)
3	＊，／
4	¥
5	MOD
6	＋，－

〔例 1〕

```
10  X = 12.3 : Y = 45.6
20  A = X + Y : PRINT A  ──→ 57.9が出力される
30  B = X − Y : PRINT B  ──→ −33.3
40  C = X * Y : PRINT C  ──→ 560.88
50  D = X/Y : PRINT D    ──→ 0.269737
```

この例は普通の4則演算（加減乗除）です．

〔例 2〕

```
10  A = 123
20  X = −A
30  PRINT X  ──→ −123 が出力される
```

この例の20番の代入文の右辺では，記号−がマイナス符号として使われています．この代入文は次のように書いたものと同じ結果になります．

```
20  X=−1*A  あるいは  20  X=A*(−1)
```

〔例 3〕

```
10  A = 2 : B = 3
20  PRINT A∧B  ──→ 8が出力される．(A^B = 2^3 = 8)
30  PRINT A∧(−2)  ──→ 0.25 (A^-2 = 1/A^2 = 1/4)
40  PRINT A∧0.5  ──→ 1.41421 (A^0.5 = √A)
50  X = (−A)∧0.5  ──→ エラー(Illegal function call) ……√−2 = 虚数
```

この例のように，演算子∧はベキ乗を意味します． BASICの数値演算は（数学的意味の）実数の範囲のみを扱うので，$\sqrt{-2}$という演算は行なわれません．

〔例 4〕

```
10  A = 27 : B = 4
20  X1 = A¥B : PRINT X1  ──→ 6が出力される．(A÷B = 27÷4)
30  Y1 = A MOD B : PRINT Y1  ──→ 3 (A÷Bの余り)
40  W1 = A MOD 9 : PRINT W1  ──→ 0 (27÷9の余り→割り切れる)
50  X2 = 27.6¥4.8 : PRINT X2  ──→ 6 (27÷4)
60  Y2 = 27.6 MOD 4.8 : PRINT Y2  ──→ 3 (27÷4の余り)
70  W2 = −27.6 MOD 4.8 : PRINT W2  ──→ 0 ((−28)÷4の余り→割り切れる)
```

この例では演算子¥およびMODを示しています．

¥は整数扱いの除算であり，結果は整数値となります．MODは整数扱いの除算を行なったときの剰余（余り）であり，結果は整数値となります．

¥あるいはMODの左右にあるデータは，自動的に整数値に変換されてから除算に用いられます．このデータの値は（−32768〜32767）の範囲になければなりません．

27 ÷ 4 ⟶ 6　　　余り　3

↑ 27¥4　　　↑ 27 MOD 4

〔例5〕
```
10  A = 4+3/2−1 ⟶ Aの値は4.5となる.(4+1.5−1)
20  B = (4+3)/(2−1) ⟶ Bの値は7となる.(7/1)
30  X = (−B+SQR(B∧2−4*A*C))/(2*A)
```

この例では演算の順序やカッコの使い方を示しています.

10番の代入文では,演算の優先順位の規則により,3/2という除算が先に行なわれます.20番の代入文ではカッコを用いているので,順序が異なります.30番の代入文を数学の式で書くと次のようになります.

$$x=\frac{-b+\sqrt{b^2-4ac}}{2a}$$

〔例6〕
```
10  A#=5#/7:PRINT A# ⟶ 0.7142857142857143
20  B=5#/7:PRINT B ⟶ 0.714286
30  C#=5/7:PRINT C# ⟶ 0.7142857313156128
```

算術式の値(演算結果)の型(整数型,単精度型,倍精度型)は,式の中にある最高精度のデータの型と同じになります.したがって,5#/7という式の値は,5#と同じ型(倍精度型)となります.

20番の代入文では右辺の式の値は倍精度型で得られますが,左辺の変数(B)が単精度型なので,代入するときに単精度型に変換されます.

30番では,右辺の式の値は単精度型です.C#の値が長い桁数で出力されますが,その内の8桁目以降は誤差が含まれたものとみなさなければなりません.

5・4　関　数

ある特定の演算や処理を行ないその結果を求めるのに,関数を使うことができます.関数は式を構成する要素となりますが,関数それ自身だけではステートメントにはなりません.

BASICでは各種の関数があらかじめ用意されており,それを自由に呼び出して利用できるようになっています.これを組込み関数と呼びます.また,組込み関数以外の関数を使用者が自分で作って使うこともできます.

組込み関数には,数値データを処理するものや文字データを処理するもの,あるいは特殊な働きをするものなどいろいろありますが,本節では5・4・1で,数値データの演算に関するものを紹介します.文字データに関する組込み関数は第6章で示します.また,自分で関数を作る方法を5・4・2で説明します.

一般的にいって,関数は次のような形をしています.

<関数名>（<引数>）

例えば，SQR（X）はXの平方根を求める関数ですが，SQRが関数名です．関数名の右のカッコの中に書かれるものを引数と呼びます．引数としては一般に，変数名，定数あるいは式を書くことができます．引数の型や値の範囲は各関数ごとにきめられており，それに適合しない引数を用いるとエラーとなります．例えばXにマイナスの値を入れてSQR（X）を使うと，エラー（Illegal function call）となります．

5·4·1　組込み関数

本項で紹介する組込み関数は次のものです．それぞれについて順に説明します．

- 数値の基礎的な処理………絶対値（ABS），符号（SGN），小数部分の丸め（FIX, INT）
- 数学の初等関数…………
  - 平方根（SQR）
  - 対数関数（LOG），指数関数（EXP）
  - 三角関数（SIN, COS, TAN），逆三角関数（ATN）
- 数値の型の変換……………整数型（CINT），単精度型（CSNG），倍精度型（CDBL）
- 乱数の発生…………………一様乱数（RND）

ABS(X) ……Xの絶対値，数学の記号では $|X|$

〔例1〕 ABS(2－5) ──→関数の値は3となります．ABS(0) ──→ 0となります．

SGN(X) ……Xの符号を求める．

この関数の値は，X＞0ならば1，X＜0ならば－1，X＝0ならば0となります．

〔例2〕 SGN(12.3)──→ 1， SGN(－12.3) ──→－1， SGN(0) ──→ 0

FIX(X) ……Xの小数部分を切捨てる．

〔例3〕 FIX(2.78) ──→ 2，FIX(－2.78) ──→ －2

この例でわかるように，関数 FIXの働きは4 捨5入ではありません．切捨てです．

INT(X) ……「Xより大きくない整数」の中の最大のもの．

X≧0のときはINT(X) と FIX(X) とは同じ結果になりますが，X＜0のときは異なります．

〔例4〕 INT(2.78) ──→ 2，INT(−2.78) ──→ −3………要注意
INT(32768.123) ──→ 32768

SQR(X) ……Xの正の平方根$\sqrt{X}$，(X≧0)

X≧0という制限に注意してください．例えば，SQR(−2) ──→ エラーとなります．

〔例5〕
```
10 PRINT SQR(2) ──→ 1.41421
20 PRINT SQR(2#) ──→ 1.41421
```
} 同じ結果が出力されます．

このように，引数Xを倍精度型にしても関数の値は単精度となります．

LOG(X) ……自然対数 $\log_e X$，(X＞0)

X＞0ということ，また常用対数（$\log_{10} X$）ではなく自然対数であることに注意してください．

〔例6〕 LOG(10) ──→ 2.30259 LOG(1) ──→ 0，LOG(0) ──→ エラー

EXP(X) ……指数関数 e^x (X＜87.3366)

〔例7〕 EXP(1) ──→ 2.71828，EXP(−1) ──→ .367879，EXP(0) ──→ 1
EXP(87.3386) ──→ エラー (Overflow)

SIN(X) ……三角関数 sinX，(Xはラジアン)

COS(X) ……三角関数 cos X，(Xはラジアン)

TAN(X) ……三角関数 tan X，(Xはラジアン)

上記3種の三角関数の引数Xはいずれもラジアンです．また関数の値は単精度で与えられます．

〔例 8〕

```
10 INPUT D ←── Dに30（度）を入力すると,
20 X=D/180＊3.14159 ……D（度）をX（ラジアン）に変換する.
30 PRINT SIN(X);COS(X);TAN(X) ──→ .5, .866026, .57735 が出力される.
```

ATN(X) ……逆三角関数 arctan X ($\tan^{-1}$X), (ATN(X)の値はラジアン)

この関数の値は，次の範囲のラジアンで与えられます．$-\pi/2 < \mathrm{ATN}(X) < \pi/2$.

〔例 9〕 ATN(1) ──→ .785398（ラジアン）, ATN(0) ──→ 0

CINT(X) ……Xを整数型に変換する.

Xの小数部分を丸めて，整数型のデータに変換します．小数部分の丸め方は前掲のINTの場合と同じです．関数CINTの値は整数型ですから，その値の範囲は−32768〜32767に限られます．したがって，Xの小数部分を丸めた結果がこの値の範囲を超えていれば，整数型として表現できず，エラー（Overflow）となります．この点が前掲のINT関数とは異なります．

〔例10〕 CINT(2.78) ──→ 2, CINT(−2.78) ──→ −3……要注意
CINT(32768.123) ──→ エラー(Overflow)……〔例 4〕とくらべてください.

CSNG(X) ……Xを単精度型に変換する.

〔例11〕 CSNG(1234.56789) ──→ 1234.57

CDBL(X) ……Xを倍精度型に変換する.

〔例12〕 CDBL(1234.57) ──→ 1234.570068359375

RND(X) ……乱数を発生する.

この関数は，0と1の間の一様乱数を発生します．引数Xは普通は任意の正の値を使います．Xを0にすると，前回と同じ値が何回でも続きます．

〔例13〕

```
10 FOR J＝1  TO  5
20 S=INT (RND (1) *6+1) : PRINT S ──→ (例えば)2, 5, 3, 6, 4 が出力される.
30 NEXT J
```

これを実行すると，1～6の範囲の整数値を等確率でとる乱数（サイコロの目）を5個発生します．

5·4·2 自分で作る関数

前項で述べた組込み関数の他に，使用者が自分で関数を作って用いることができます．そのためには，次に示す関数定義文（DEFFN文）をプログラムの中にあらかじめ書いておくことが必要です．

DEFFN 〈名前〉（〈引数〉）＝〈式〉……関数定義文

DEFFNの右に書く名前は関数を識別するためのものです．名前のつけ方の規則は，普通の変数名の命名の規則と同じです．第2章（2·2·2）を参照してください．

引数は変数名を使ってカッコの中に書きます．引数が複数個のときはコンマ(,)で区切って書きます．引数は全然無くてもよろしい．その場合はカッコとともに省略します．

また，引数を表わす変数名は，プログラム中にある他の変数の名前と同一でもかまいません．たとえ同一名であっても，それは無関係なものとみなされ，互いに影響は受けません．

右辺の式は関数の演算内容を示すものです．この式の中の変数名は，引数と同じものだけでなく，その他の変数名でもよろしい．

関数定義文によって定義された関数を呼び出して使うときの要領は，組込み関数のときと同様です．その場合，関数名として「FN〈名前〉」を用います．つまりFNを頭に必ず付けます．FNが付いていると，それは使用者が作った関数であると解釈されるのです．

FN 〈名前〉（〈引数〉）……使用者定義関数の呼び出し

関数が呼び出されると，関数定義文の中の引数(仮引数という)は，関数呼び出しの中の引数（実引数という）に置き替えられます．仮引数は必ず変数名で書きますが，実引数は変数名でも定数でも式でもよろしい．実引数の変数あるいは値が，仮引数と順に1対1に対応して置き替えられた形で,式の演算が行なわれて関数の値が与えられます．

なお，定義する関数は数値型でも文字型でもよろしい．ただし，関数の名前の型と右辺の式の型とは一致していなければなりません．

〔例 1〕

```
10  DEFFN R(X, Y)=SQR (X*X+Y*Y)
        ⌇
100  INPUT A, B  ←— A=3, B=4を入力すると
110  C=FNR (A, B)  —→ Cの値は5となる
120  D=FNR (3, 4)  —→ Dの値も5となる
```

この例では10番が関数定義文です．名前はRであり，引数はXとYの2個です．式は $\sqrt{X^2+Y^2}$ という内容です．110 番の代入文の中で関数を呼び出しています．実引数はAとBであり，X，YをA，Bに置き替えた形で演算されます．120 番の代入文の中の関数呼び出しでは，実引数として定数が使われています．

SQR(3*3+4*4) —→ $\sqrt{25}$ —→ 5

〔例 2〕

```
10  DEFFNF 1 (X)=A*X*X+B*X+C
        ⌇
100  INPUT A, B, C
200  INPUT X
210  Y=FNF 1 (X)
```

これは数学の式で書けば，$f_1(x)=ax^2+bx+c$ という関数を定義したものです．関数定義文の式の中のA，B，Cは引数とは別の変数であり，100番のINPUT文で入力された値が演算に使われます．A，B，Cに1，2，3を入力し，Xに4を入力すれば，Yの値は27となります．

〔例 3〕

```
10  DEFFN S=A+B+C
20  INPUT A, B, C
30  Y=FNS*5+2
```

これは引数なしの関数の例です．A，B，Cに1，2，3を入力すれば，Yの値は32となります．

〔例 4〕

```
10  DEFFN P$=CHR$ (12)+"NEXT"
        ⌇
100  PRINT FNP$
```

関数定義文の名前に$が付いているので，文字型の関数です．右辺の式も文字式です．100 番のPRINT文を実行すると画面が消去されて，最上行にNEXTと表示されます．なお，第6章（6.1·2および6·4·1）を参照してください．

理解度テスト

(1)　次の数式の計算を，算術式で書きなさい．

①　$\dfrac{x-2}{y+4}$　　②　$\dfrac{bc}{a\,(1+b^2)}$　　③　$\left(\dfrac{x}{y}\right)^2$　　④　nx^{n-1}　　⑤　$\sqrt{1+x^2}$

(2)　次のものは，代入文として文法的に正しいか．正しいものには○をつけ，誤りのものには×をつけなさい．

①　A＋B＝C　　　　⑥　LC＝LC＋1
②　LET　X＝Y　　　⑦　M＝A＄
③　\＝B　　　　　　⑧　A＝B＋B
④　A＝－A　　　　　⑨　A%＝3.14159
⑤　X　AND　Y＝0　⑩　A%＝B#＋C！

(3)　次の指数型定数を，基本型定数の形に書きなおしなさい．

①　－5.4E2　　②　0.5E－1　　③　123.4E0　　④　123.4D1　　⑤　－5D－3

(4)　次の算術式の値はいくらか，答えなさい．

①　－2∧3　　　　②　－2∧4　　　　③　2∧2∧2　　　　④　(－2)∧3∧2
⑤　365￥7　　　　⑥　－365￥7　　　⑦　123.4￥123.4
⑧　365　MOD7　　⑨　12 MOD 34　　⑩　123.4　MOD 123.4

(5)　X＝－0.5 のとき，次の関数の値はいくらになるか，答えなさい．

①　INT(X)　　②　SGN(X)　　③　FIX(X)　④　ABS(X)　　⑤　CINT(X)

(6)　次の文のうち，内容が正しいものには○をつけ，誤りのものには×をつけなさい．

①　整数型データの値は，0（ゼロ）より小さくなることは許されない．
②　整数型変数の値としては，32767 より大きいものは許されない．
③　何も宣言しないで変数を使うと，その変数は自動的に整数型とみなされる．
④　数値定数の末尾にDという文字を付けると，倍精度型とみなされる．
⑤　数値定数の末尾に型宣言文字が無ければ，自動的に単精度型とみなされる．

第６章　文字データの処理

コンピュータは数値データの処理をするだけでなく，文字もデータとして処理することができま

ず．文字データの処理というのは，入力，出力，比較，分解，合成，変換その他です．このうち，入力については第3章で述べました．また出力については第8章以降で述べます．本章では，各種の関数を利用するなどして文字データを取扱う方法を紹介します．

プログラムの中では，文字データは，文字定数あるいは文字変数として取扱われます．これについて，ここで基礎的なことを示しておきます．

① 文字定数は，何個かの文字を並べて構成します．文字列とかストリングともいいます．1個の文字定数の長さは255文字以内でなければなりません．また，長さ0（ゼロ）の文字列も文字定数とみなします．これをヌルストリングといいます．

プログラムの中に文字定数を書くときは，それを引用符（″）で囲むのが原則です．ただし例外として，引用符が不要な場合もあります．例外についてはそのつど示します．

② 文字変数は，文字データを記憶させる場所であり，変数名によって表わされます．文字変数であることを示すために，変数名の末尾にドル記号（$）を付けます．あるいは型宣言文によって，文字変数であることを宣言することもできます．これらについては，第2章(2·2·2)および第5章(5·1·1)を参照してください．

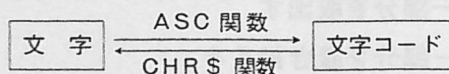
6·1　文字～文字コードの変換

BASICでプログラムを書いたりデータの値を表現したりするために，いろいろな文字を使います．それらの各文字に対して，「文字コード」というものが定められています．文字コードとは，それぞれの文字に付けられた番号だと考えることができます．コンピュータがその内部で文字を取扱うときは，文字の形（字形）そのものを取扱うのではなく，文字コードとして記憶したり処理したりしているのです．

例えば「A」という文字の文字コードは65（10進法）です．「B」は66，「a」は97，「ア」は177，「#」は35です．これらのすべてを文字コード表に掲げておきます．

プログラムの中でいろいろな処理をする場合に，ある文字のコードを求めたり，あるいは逆に文字コードに対応する文字を求めたりすることが必要になります．例えば，文字をコードに変換すれば数値として大小を比較することができ，それによって文字データをアルファベット順に並べ替えることが可能になります．また文字の中にはある特定の働きを意味するもの（機能文字）があり，これを取扱うときには文字コードを文字に変換して用いることが必要になります．(例えば画面消去という働きを意味する文字のコードは12)

文字と文字コードとの相互間の変換には次のような関数を使いますが，その詳細は以下の各項で解説します．

文　字　→（ASC 関数）→　文字コード
文　字　←（CHR$ 関数）←　文字コード

ところで，「A」の文字コードは65（10進法）である，と上で述べましたが，これを16進法で表わすと41となります．また2進法で表わせば01000001となります．このことに関して，ここで説明を加えておきましょう．（初心者の方は読み飛ばして結構です）

コンピュータの内部では，1文字を1バイト（すなわち8ビット）で表わしています．いいかえれ

ば8桁の2進数で表わしています．各文字にはそれぞれ対応する2進数が定められており，これがその文字の文字コードなのです．

8桁の2進数を中央で分割した左側を「上位4ビット」と呼び，右側を「下位4ビット」と呼びます．上例の文字「A」の場合，上位4ビットは0100です．0100（2進法）＝4（16進法）です．また下位4ビットは0001です．0001（2進法）＝1(16進法です．この4と1とを組合わせて41とすると，これが16進法による表現となります．

またカナ文字「カ」(文字コードは10進法で182)について，上と同様に考えると次のようになります．

```
 1  0  1  1    0  1  1  0 ←―― 2進法
 └────┬────┘   └────┬────┘
  上位4ビット     下位4ビット
      ‖              ‖
   11(10進)       6(10進)
      ‖              ‖
   B(16進)        6(16進) ――→文字「カ」のコードは，B6(16進法)
```

上述のことからわかるように，文字コードの値は10進法で0～255の範囲にあります．16進法ならば，0～FFの範囲です．

次に掲げる文字コード表は，16進法で索引するようにできています．例えば上位4ビットが「B」の例と，下位4ビットが「6」の行の交点に，文字「カ」があります．この表は10進法で索引するには不便ですが，しかし文字コードがどのように割当てられているかを見るには役立ちます．例えば，英大文字と英小文字とのコードの関係は，小文字コード＝大文字コード＋32（10進法）となっていることがわかります．また上位4ビットが0の列と1の列とにある文字は，特定の働きを意味する「機能文字」です．

〔文字コード表〕（16進法）

上位4ビット→ / 下位4ビット↓

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		$\mathrm{D_E}$		0	@	P		p				ー	タ	ミ		
1	$\mathrm{S_H}$	$\mathrm{D_1}$	!	1	A	Q	a	q			。	ア	チ	ム		円
2	$\mathrm{S_X}$	$\mathrm{D_2}$	"	2	B	R	b	r			「	イ	ツ	メ		年
3	$\mathrm{E_X}$	$\mathrm{D_3}$	#	3	C	S	c	s			」	ウ	テ	モ		月
4	$\mathrm{E_T}$	$\mathrm{D_4}$	$	4	D	T	d	t			、	エ	ト	ヤ		日
5	$\mathrm{E_Q}$	$\mathrm{N_K}$	%	5	E	U	e	u			・	オ	ナ	ユ		時
6	$\mathrm{A_K}$	$\mathrm{S_N}$	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		分
7	$\mathrm{B_L}$	$\mathrm{E_B}$	'	7	G	W	g	w			ァ	キ	ヌ	ラ		秒
8	$\mathrm{B_S}$	$\mathrm{C_N}$	(	8	H	X	h	x			ィ	ク	ネ	リ	♠	
9	$\mathrm{H_T}$	$\mathrm{E_M}$	)	9	I	Y	i	y			ゥ	ケ	ノ	ル	♥	
A	$\mathrm{L_F}$	$\mathrm{S_B}$	*	:	J	Z	j	z			ェ	コ	ハ	レ	♦	
B	$\mathrm{H_M}$	$\mathrm{E_C}$	+	;	K	[	k	{			ォ	サ	ヒ	ロ	♣	
C	$\mathrm{C_L}$	→	,	<	L	¥	l	¦			ャ	シ	フ	ワ	●	
D	$\mathrm{C_R}$	←	−	=	M	]	m	}			ュ	ス	ヘ	ン	○	
E	$\mathrm{S_O}$	↑	.	>	N	^	n	~			ョ	セ	ホ	・		
F	$\mathrm{S_I}$	↓	/	?	O	_	o				ッ	ソ	マ			

6·1·1　文字から文字コードへの変換

前述のように，BASICで使用する各文字にはそれぞれ文字コードが定まっています．次の関数

```
ASC (X$)
```

ASCを使えばこのコードを求められます．

カッコの中のX＄は文字型の引数です．文字型変数名でもよいし，引用符（″）で囲まれた文字列（文字定数）でもよろしい．その文字データの「最初の１文字」の文字コードが，関数ＡＳＣの値となります．この値は，10進法の数値として与えられます．

〔例１〕

```
10  M$ ="t"
20  X 1=ASC (M$ )    ←──文字変数M$ の値(t)のコードを求める.
30  X 2=ASC ("T")    ←──文字(T)のコードを求める.
40  X 3=ASC ("TOKYO") ←──最初の1文字(T)のコードを求める.
50  PRINT  X 1; X 2; X 3
```

Ｘ１の値は116，Ｘ２およびＸ３の値はいずれも84となります.（文字コード表を見てください）

〔例２〕

```
10  N $ =""        ←──引用符（ ″ ）を続けて(間を空けずに) 2個書く.
20  X =ASC (N$)    ──→エラーとなる
30  PRINT  X
```

この例では文字データ（Ｎ＄）の値がヌルストリング（長さ０の文字列）です．ヌルストリングに対応する文字コードは求められず，エラー（Illegal function call）となります．

6·1·2　文字コードから文字への変換

次の関数ＣＨＲ＄は前項のＡＳＣの逆の働きをします．すなわち，文字コード（10進法）に対応する文字を求めます．

```
CHR$ (J)
```

カッコの中のＪは数値型の引数です．数値定数，変数名または数値式で書きます．Ｊは文字コードを表わすものですから，その値は０～255の範囲になければなりません．またＪの値が整数でない場合は，小数部分は自動的に切捨てられます．

〔例 1〕

```
10  N＝84：A$ ＝CHR$ (N)：PRINT  A$  ──→文字T が出力される.
20  PRINT  CHR$ (N＋0.5)  ──────→ 文字T が出力される.
30  PRINT  CHR$ (284)  ──→エラー(Illegal function call) となる. (284＞255)
```

〔例 2〕

```
10  PRINT  CHR$ (12);   ──→画面を消去する.
20  PRINT  CHR$ (31);   ──→カーソルが 1 行下へ動く.
30  PRINT  CHR$ (28);   ──→カーソルが 1 桁右へ動く.
40  PRINT  "ABC"
50  PRINT  CHR$ (34)    ──→引用符（"）を出力する.
60  PRINT  CHR$ ( 7 )   ──→ブザーを鳴らす.
```

画面の最上行→

	A	B	C		
"					
O	K_				

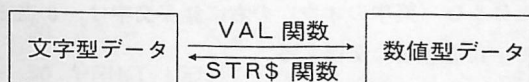
機能文字や引用符などは，この例のようにＣＨＲ＄を使えば出力することができ，機能文字はその働きを実行します．画面には図のように表示されます．

6・2　文字データ～数値データの変換

BASICでは数値でも文字でもデータとして取扱うことができますが，数値データと文字データとではコンピュータ内部での記憶の形式や処理の仕方が異っています．したがって両者をむやみに混同して使うことはできません．例えば，

```
10  A＝123：B＄＝"123"：X＝A＋B＄
```

というようなことはできません．見かけ上は似ていても，Ａは数値型，Ｂ＄は文字型ですから，Ａ＋Ｂ＄という式はエラー（Type mismatch）となります．123＋123→246 という結果は得られません．そこで場合によっては，数値データと文字データとを相互に変換する必要が生じます．この変換のための関数としてＶＡＬとＳＴＲ＄があるので，以下に解説します．

［文字型データ］ ──VAL 関数──→ ［数値型データ］
［文字型データ］ ←──STR$ 関数── ［数値型データ］

6・2・1　文字型から数値型への変換

次の関数ＶＡＬは文字型データを数値型データに変換する働きをします．

```
VAL (X$)
```

カッコの中のＸ＄は文字型の引数です．文字型変数名でもよいし，引用符（"）で囲まれた文字列（文字定数）でもよろしい．その文字データが表現している内容を数値データに変換したものが，

関数ＶＡＬの値となります．この値は10進法で与えられます．

変換される文字列の最初の文字が「＋，－，＆，空白，または数字」でなければ，関数ＶＡＬの値は０（ゼロ）となります．また文字列の中に，数値に変換できないような文字があると，それ以降の文字が無視されるか，あるいはエラーとなります．

〔例１〕　20　A＝123：B$＝"123"：X＝A＋VAL(B$)

ＶＡＬ(Ｂ＄)の値は123という数値型になります．したがってＡとの加算が可能であり，Ｘの値は246となります．

〔例２〕　30　INPUT A$：PRINT VAL(A$)

Ａ＄に入力した文字列を数値型データに変換したものが，下記のように画面に出力されます．

（Ａ＄への入力）　（画面への出力）
0123456789 ──→123456789
－123.45 ──→－123.45
A 123.45 ──→ 0 ……………最初の文字がＡなので，VALの値は０（ゼロ）となる．
123AB45 ──→123 ……………文字Ａ以降は無視されている．
1.2345E 2 ──→123.45…………E 2 は×10^2と解釈されている．
␣1.23E－2 ──→.0123 …………第１文字目の␣は空白．

〔例３〕 上例と同じステートメントを用いて，最初の文字が＆になっている文字列を入力すると，次のようになります．（16進法あるいは８進法の表現とみなされます．第５章（5･1･2）参照）

（A$への出力）（画面への出力）
＆H100 ──→256 ⎫
＆HFF ──→255 ⎭……… ＆Ｈの右に並ぶ文字は，16進法の表現とみなして，10進数値に変換する．
＆100 ──→64 ⎫
＆O17 ──→15 ⎭……… ＆または＆Ｏ（英字のオウ）の右に並ぶ文字は，８進法の表現とみなして，10進数値に変換する．
＆FF ──→ 0
＆9 ──→Syntax error

6･2･2　数値型から文字型への変換

次の関数ＳＴＲ＄は前項のＶＡＬの逆の働きをします．すなわち，数値を文字列に変換します．

STR$　(J)

カッコの中のＪは数値型の引数です．数値定数，変数名または数値式で書きます．

〔例〕　10 INPUT X： A$＝STR$ (X) ： PRINT A$; LEN (A$)

Ｘに入力した数値を文字列に変換したものがＡ＄の値となります．またLENは文字列の長さ(字数）を求める関数です．(次節6·3参照)

(Ｘへの入力)		(A$ の値)		(LEN(A$))
−123	⟶	−123	⟶	4
＋123	⟶	␣123	⟶	4
123.45	⟶	␣123.45	⟶	7
1.2345E 2	⟶	␣123.45	⟶	7
0	⟶	␣0	⟶	2

変換によって作られた文字列の長さに注意してください．この例の２番目以降では，文字列の最初の文字は空白となります．このように符号を表わす分として１文字が使われます．

6·3　文字列の長さを求める

文字データを処理する場合，その長さ（文字列の字数）を知る必要が起こります．このために次の関数ＬＥＮがあります．

```
LEN　(X$)
```

カッコの中のＸ＄は文字型の引数です．文字型変数名でもよいし，引用符（″）で囲まれた文字列でもよろしい．その文字データの長さが，関数ＬＥＮの値（０～255の範囲)となります．

文字列の中に空白や特殊な文字（機能文字）が含まれていれば，それもそれぞれ１文字と数えます．またヌルストリング(文字を含まない文字列)の長さは０（ゼロ）となります．

〔例〕

```
10  X 1＝LEN(" キンガク")        ──→X 1の値は5となる.
20  A$＝"" : X 2＝LEN(A$)        ──→X 2の値は0となる.(ヌルストリング)
30  PRINT LEN("I␣love␣you.")    ──→11が出力される.(空白も数える)
40  X 3＝LEN(STR$ (123))         ──→X 3の値は4となる. (6·2·2参照)
50  X 4＝LEN(CHR$ (12))          ──→X 4の値は1となる. (6·1·2〔例 2〕参照)
```

6·4　文字列の合成

複数個の文字データを連結して１個の文字データにまとめることができます．また，同じ文字を複数個並べて１個の文字列とすることもできます．すなわちプログラムの中で，文字列を合成して処理に利用することができます．ただし，合成されてできた文字列の長さは255字以内でなければなりません．

文字列を合成する方法として，次のものを以下

の各項で説明します.

① 加算記号（＋）で文字データを連結する.（6·4·1参照）

② 関数STRING＄を使って同一の文字の並びを作る.（6·4·2参照）

③ 関数SPACE＄を使って空白の並びを作る.（6·4·3参照）

6·4·1 文字データの連結

次のように加算記号（＋）を使うことによって，2個以上の文字データを連結して1個の文字データを作り出すことができます.

〈文字データ〉＋〈文字データ〉＋……

連結される各文字データとしては，引用符(″)で囲まれた文字列（文字定数）や文字型変数名あるいは文字型の値を持つ関数などが使えます.

数値の計算をする式と形が似ているので，上記のものを「文字式」と呼びます．しかし文字式の加算記号（＋）は文字列の連結を意味するものですから，数値の加算とは違います．また文字式では，他の演算（－，＊，／など）は使えません.

〔例1〕
```
10  U$＝"ウリアゲ":K$＝"キンガク"
20  A$＝U$＋K$ :PRINT  A$      ──→ウリアゲキンガクと出力される.
30  PRINT  U$＋"スウリョウ"    ──→ウリアゲスウリョウと出力される.
```

〔例2〕
```
1   Q$＝CHR$(34)
20  PRINT  CHR$(12)＋Q$＋"START"＋Q$
```

画面の最上行──→ "START"

この例では，関数CHR＄と他の文字データとを連結しています．これによって右図のように画面に出力されます．34は引用符（″）の文字コード，12は画面消去の文字コードです.（6·1·2参照）

6·4·2 同一文字の並びを作る

同じ文字をいくつか並べて1個の文字列を作るために，次の関数STRING＄があります．これを使えば，どんな文字を何個並べるのか，を引数によってそのつど指定できるので便利です．この関数には，文字の指定の仕方によって2種類の形があります.

STRING$ (N, X$)

……文字データX＄の最初の1文字をN個並べる.

STRING$ (N, J)

……文字コードJに対応する文字をN個並べる.

カッコの中の引数Nは，作るべき文字列の長さ（字数）を指定するものです．数値型の定数，変数名あるいは数値式を書きます．Nの値は255を越えることはできません．またNの値が整数でな

いときは，小数部分は自動的に切捨てられます．

引数X＄は文字型です．引用符（”）で囲まれた文字列あるいは文字型変数名あるいは文字型の値を持つ関数などを書きます．引数X＄の最初の文字が，並べるべき文字として指定されます．

引数Jは数値型です．これは並べるべき文字の文字コード（10進数）を指定するものであり，CHR＄関数の引数と同様なものです．（6·1·2参照）

引数X＄またはJによって指定された文字をN個並べた文字列が，関数STRING＄の値として与えられます．

〔例1〕

```
10  INPUT  N, X$  ←―――――――― Nに3，X$ にAを入力すると，
20  PRINT  STRING$ (N,  X$)  ――――→AAA が出力される．
30  PRINT  STRING$ (4， "PQR")  ――→PPPP が出力される．
40  PRINT  STRING$ (N, CHR$(7))――→ブザーが3回続けて鳴る．
```

〔例2〕

```
50  INPUT  N, J  ←―――――――― Nに3，Jに65を入力すると，
60   PRINT  STRING$ (N，J)  ――――→AAA が出力される．
70   PRINT  STRING$ (N，J＋32) ――→aaa が出力される．
80   PRINT  STRING$ (2＋N，65)  ――→AAAAA が出力される．
```

〔例3〕

```
10  FOR  J＝1   TO  5
20  A$ ＝STRING$ (J, "*")
30  PRINT  A$
40  NEXT  J
```

```
*
**
***
****
*****
```

これを実行すると右図のように画面に出力されます．棒グラフを描くときなどに利用できます．

6·4·3　空白の並びを作る

次の関数SPACE＄は，指定された個数の空白を並べて1個の文字列を作ります．

```
SPACE$  (N)
```

引数Nは数値型です．並べるべき空白の個数を指定するものであり，その値は0～255の範囲になければなりません．もしNの値が整数でないときは，小数部分は自動的に切捨てられます．

SPACE＄は，前項のSTRING＄において文字として空白を指定したものと同じ結果になります．

〔例〕

```
10  FOR  J＝0   TO  5
20  X$ ＝SPACE$ (J)
30  PRINT  X$; CHR$ (J＋65)
40  NEXT  J
```

```
A
 B
  C
   D
    E
     F
```

6·5　文字列の分解

本節で紹介するのは，文字列の中の一部分を取扱う方法です．すなわち，文字列を分解して一部分だけを取出したり，また一部分を他の文字で置き換えたりすること，あるいは指定した文字が含まれているかどうかを調べることなどです．

① 一部分を取出す．(6·5·1参照)

② 一部分を置き換える．(6·5·2参照)

③ 含まれている文字を捜す．(6·5·3参照)

このためにRIGHT＄，LEFT＄，MID＄，INSTRなどが使われます．以下の各項で解説しますが，似たようなものがいろいろあるので，注意深く読んで理解してください．

6·5·1　文字列の一部分を取出す

ある文字列の中から，指定された一部分を取出す働きをする関数があります．それはRIGHT＄，LEFT＄およびMID＄の3種です．

まずRIGHT＄とLEFT＄とを説明しましょう．この2つはよく似ており，右か左かという点が違うだけです．

RIGHT$　(X$, N)……右端から．

LEFT$　(X$, N)……左端から．

引数X＄は文字型です．Nは数値型の引数であり，X＄の中から取出すべき文字の個数を指定するものです．Nの値は0～255の範囲になければなりません．

X＄の右端からN文字を取出したものが関数RIGHT＄の値として与えられます．同様に左端からN文字分がLEFT＄の値となります．

N＝0の場合は，RIGHT＄やLEFT＄の値はヌルストリング（長さ0の文字列）となります．

X＄の長さ（すなわちLEN(X＄)）よりもNの方が大きい場合は，RIGHT＄やLEFT＄の値はX＄そのものと等しくなります．

〔例1〕

```
10 X$＝"ABCDE"
20 PRINT RIGHT$ (X$, 3)────→CDE が出力される.
30 PRINT LEFT$ (X$, 5) ────→ABCDE が出力される.
40 PRINT LEFT$("ABC", 5) ──→ABC が出力される.
50 A$=RIGHT$ ("ABC", 0)────→A$ はヌルストリングとなる.
```

では次にMID＄を紹介します．これは文字列の中程の部分を取出すことができます．

MID$　(X$, J, N)

引数X＄およびNの意味は，上述のRIGHT$やLEFT＄の場合と同じです．Jは数値型の引数であり，X＄の中のどの位置から文字を取出し始めるのかを指定するものです．Jの値は1～255の範囲になければなりません．

X＄の左端から数えてJ番目の文字から始まるN個の文字を取出したものが関数MID＄の値となります．

X＄の長さ（LEN(X$)）よりもJの方が大きい場合は，MID＄の値はヌルストリングとなります．

また，J番目の文字から右方に，N個より少ない文字しかない場合は，そこにある分だけがMID$の値となります

〔例2〕

```
10 X$="ABCDE"
20 PRINT MID$(X$, 3, 2) ────→CD が出力される.
30 PRINT MID$(X$, 3, 4) ────→CDE が出力される.
40 A$=MID$(X$, 7, 1)    ────→A$ はヌルストリングとなる.
```

また，MID＄の第3番目の引数Nを書かない形もあります．次のようになります．

```
MID$ (X$, J)
```

この場合は，J番目から右方にあるすべての文字が取出されて，それがMID＄の値となります．

〔例3〕 MID$（"ABCDE"，2）の値は，BCDEとなります．

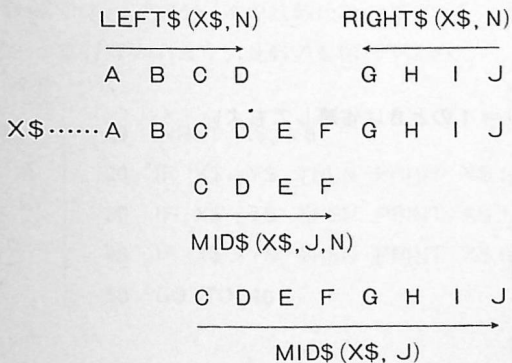
以上に紹介した各関数を整理して比較してみましょう．

X＄＝"ABCDEFGHIJ"：N＝4：J＝3とすると次図のようになります．

```
          LEFT$(X$,N)       RIGHT$(X$,N)
          ──────→           ←──────
          A  B  C  D        G  H  I  J

X$……A  B  C  D  E  F  G  H  I  J

                C  D  E  F
                ──────→
                MID$(X$,J,N)

                C  D  E  F  G  H  I  J
                ──────────────→
                     MID$(X$, J)
```

6·5·2 文字列の一部分を置き換える

文字列の中の指定された部分を他の文字に置き換えるには，次の形が使えます，

```
MID$ (X$, J, N)=Y$
```

これは前項のMID＄に右辺「＝Y＄」を付け加えた形をしています．左辺のカッコの中の各引数

(X$, J, N）の意味は，前項で述べたものと同様です．右辺のY$は文字型データです．

「X$の左端から数えてJ番目の文字から始まるN個の文字」を「Y$の左端からN文字」で置き換えます．これを実行すると，X$の内容は変化して，一部分が置き替えられた文字列になります．

〔例1〕

```
10  A$="123456789":B$="PQRST"
20  MID$(A$, 5, 3)=B$
30  PRINT  A$ ──────→ 1234PQR89 と出力されます.
```

また，第3番目の引数Nを書かない形もあります．次のようになります．

```
MID$ (X$, J)=Y$
```

〔例2〕

```
10  A$="123456789"
20  MID$(A$, 3)="PQRST":PRINT  A$  ──→ 12PQRST89 と出力される.
30  B$="123456789"
40  MID$(B$, 7)="PQRST":PRINT  B$ ──→ 123456PQR と出力される.
```

6·5·3　文字列の中の文字を捜す

次の関数INSTRは，文字列X$の中に文字列Y$が含まれているかどうかを調べて，見つかった位置を与えるものです．

```
INSTR (J, X$, Y$)
```

```
INSTR (X$, Y$)
```
……J＝1のときは省略してもよい．

引数Jは数値型であり，X$およびY$は文字型です．Jは調べ始める位置を指定するものであり，その位置から右へ向ってX$の中を調べて行きます．そしてY$と同じ文字列が見つかると，その位置が関数INSTRの値となります．Jが省略されているときはJ＝1とみなし，左端から調べ始めます．

もしY$が見つからなかった場合，あるいはJの値がX$の長さ（LEN(X$)）より大きい場合，あるいはX$がヌルストリングの場合は，関数INSTRの値は0（ゼロ）となります．

またY$がヌルストリングの場合は，INSTRの値はJとなります．

〔例〕

```
10  X$="THIS␣IS␣A␣PEN.":Y$="IS"
20  K1=INSTR(5, X$, Y$)    ───→K1=6 となる.
30  K2=INSTR(X$, Y$)       ───→K2=3 (Jの指定を省略している)
40  K3=INSTR(5, X$, "Q")   ───→K3=0 (Qという文字は含まれていない)
50  K4=INSTR(17, X$, Y$)   ───→K4=0 (JがX$の長さより大きい)
60  K5=INSTR(5, "", Y$)    ───→K5=0 (X$がヌルストリング)
70  K6=INSTR(5, X$, "")    ───→K6=5 (Y$がヌルストリング)
```

6·6　文字データの比較

数値データと同様に文字データについても，等しい，大きい，小さいという比較ができます．比較記号は数値データの場合と同じく，「=，>，<」およびこれらの組合わせを用います．この比較記号の左側と右側に，比較されるべき文字データを書きます．文字データとしては，引用符（"）で囲まれた文字列，あるいは文字型変数名あるいは文字式などを書くことができます．

すでに6·1で述べたように，各文字には文字コードが定まっています．文字コードは0～255の範囲の数値です．文字データの比較はこの文字コードによって行なわれます．比較の要領は下記のとおりです．

1文字と1文字との比較は，文字コードの大小によって判定されます．

比較される文字列の長さが2字以上の場合は，それぞれの文字列の左端から順に1文字ずつを対応させて，その文字コードを比較して行きます．そのとき，異なった文字コードが初めて現われた時点で，その文字コードの大小によって両文字列の大小が判定されます．またそのとき，異なった文字コードが現われる前に片方の文字列が終わりになった場合は，短い方の文字列が小さいと判定されます．

ちょっとややこしいですが，下の例を見てください．

〔例〕

```
10  INPUT  X$, Y$
20  IF  X$=Y$  THEN  PRINT  X$; "="; Y$
30  IF  X$<Y$  THEN  PRINT  X$; "<"; Y$
40  IF  X$>Y$  THEN  PRINT  X$; ">"; Y$
50  GO  TO  10
```

このプログラムを実行してX＄とY＄にいろいろな文字列を入力してみると，それが比較されて次のように画面に出ます．

A＝A
A＜B……Aの文字コードは65，Bの文字コードは66.（65＜66）
A＜a……aの文字コードは97.（65＜97）
ABC＝ABC
AB＞AA　……2字目（BとA）の比較によって判定.
AB＞AAA　…上と同様に判定される.（AAAの方が長いが，ABの方が大きい）
AB＜ABC　…2字目まで等しいが，ABCの方が長いから大きい.

6·7　日付と時刻

PC－8001にはクロック（時計）が内蔵されており，日付や時刻を保持する機能があります．現時点におけるクロックの内容を知りたいとき，あるいは日付や時刻をクロックにセットしたいときは，DATE$およびTIME$を用います．

DATE$の内容は，次の形式の文字データとして表わされます．

```
YY／MM／DD
```

YYが年，MMが月，DDが日を表わし，それぞれ2桁の数字です．間を斜線（／）で区切ります．

電源投入時には，DATE$の値は79／01／01にセットされます．

TIME$の内容は，次の形式の文字データとして表わされます．

```
HH：MM：SS
```

HHは00～23の値で時を表わします．MMは00～59の値で分を，またSSは00～59の値で秒を表わします．いずれも2桁の数字を用い，間をコロン（：）で区切ります．

電源投入時には，TIME$の値は00：00：00にセットされます．

DATE$やTIME$の使い方を，以下に例で示します．

〔例1〕　直接モードで，次のように日付や時刻をセットしたり，表示したりできます．
これらを画面に書いて，RETURNキーを押せばよいのです．

```
DATE$＝"80／05／01" ────→日付を80年5月1日にセットする
TIME$＝"12：30：00" ────→時刻を12時30分00秒にセットする
PRINT  DATE$ ─────────→日付を画面に表示する
PRINT  TIME$ ─────────→時刻を画面に表示する
```

〔例2〕

```
10 INPUT "DATE YY/MM/DD="; A$ ←—80/05/01を入力すれば,
20 DATE$=A$ —→クロックに80年5月1日をセットする
30 PRINT DATE$ —→80/05/01を画面に出力する
40 Y$=LEFT$(DATE$, 2) —→年(80)をY$に取り出す
50 M$=MID$(DATE$, 4, 2)—→月(05)をM$に取り出す
60 D$=RIGHT$(DATE$, 2) —→日(01)をD$に取り出す
70 PRINT Y$"年"M$"月" D$"日" —→80年05月01日を画面に出力する
```

この例では日付を取扱っていますが，時刻についても同じ要領でできます．DATE$もTIME$も，その値は「文字データ」として与えられることに注意してください．

理解度テスト

(1) 次のステートメントの書き方は，文法的に正しいか．正しければ○をつけ，誤りならば×をつけなさい．

① A$="A"

② A=B$+C$

③ X$=ASC(A$)

④ Y=ASC("ABC")

⑤ PRINT CHR$(265)

⑥ PRINT STRING$(5, 67)

⑦ A$=VAL(B$)

⑧ S$=STR$(A)

⑨ A$=RIGHT$(B$)

⑩ L=1+LEN(A$)

(2) 下記の左列の関数はどんな働きをするものか．右列の中からひとつずつ選んで答えなさい．

① ASC(A$)	① 文字列の長さを求める.
② CHR$(A)	② 数値型から文字型への変換.
③ INSTR(A$, B$)	③ 文字コードを求める.
④ LEFT$(A$, B)	④ 文字コードを文字に変換する.
⑤ LEN(A$)	⑤ 文字列の中の文字を捜す.
⑥ RIGHT$(A$, B)	⑥ 同じ文字の並びを作る.
⑦ SPACE$(A)	⑦ 文字列の右端から文字を取り出す.
⑧ STR$(A)	⑧ 文字型から数値型への変換.
⑨ STRING$(A, B)	⑨ 空白の並びを作る.
⑩ VAL(A$)	⑩ 文字列の左端から文字を取り出す.

(3) 次のものの値はいくらになるか，答えなさい．

① VAL ("12"+"34")

② LEN ("12"+"34")

③ ASC ("34")

④ INSTR ("12", "34")

⑤ LEN (STR$ (12))

(4) 次のものの値はどんな文字列になるか，答えなさい．

① CHR$ (97)+CHR$ (98)+CHR$ (99)

② STRING$ (4, 65)

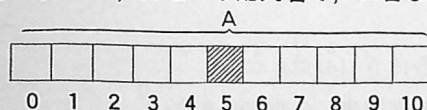
③ LEFT$ ("GFEDCBA", 2)

④ MID$ ("543210", 2, 3)

⑤ MID$ ("543210", 3)

第7章　配列による処理

配列とは，複数個の記憶場所をまとめて名前を付けたものです．この名前を配列名といいます．また配列内の個々の記憶場所を配列要素といいます．配列要素には，0（ゼロ）番から始まる一連番号が付きます．ある1個の配列要素を指すには，配列名とその要素番号とを使います．次の例を見てください．

〔例〕 次図は，Aという配列名で，10番までの配列要素を持った配列です．

└──この配列要素を指すには，A(5)と書きます．

このようにカッコ(　)で囲まれた要素番号を「添字」といいます．

すなわち配列とは，番号の付いた変数の集まりであり,「添字付き変数」とも呼ばれます．上記の記憶場所（11個）を，もし配列を使わずに表わすとすれば，A，B，C，D，E，F，G，H，I J，Kというように多くの変数名が必要です．しかし，配列を使えば，Aという1個の変数名で表わせるし，添字をうまく使えば，配列要素に対して順次に処理を行なうことも簡単にできます．同種類のデータを多数記憶しておかねばならないような処理においては，配列は不可欠な手段です．

本章ではこの配列について解説しますが，その要点は次のとおりです．

① 数値データでも文字データでも，配列を使うことができます．
② 1次元だけでなく，2次元，3次元，……の配列が使えます．
③ 配列を使うときは，まずその宣言をすることが必要です．
④ 配列を使うときは，FOR文とNEXT文を活用することが必要です．
⑤ 問題に応じて，配列をうまく利用するように心がけることが大切です．

まず7·1および7·2で，上記の①，②，③について解説します．

次に7·3で④を解説します．

また7·4では⑤のひとつの例として，データの並べ替えの方法を示します．

7·1 配列の宣言

プログラムの中で配列を使おうとするときは，それに先立ってまず，どんな配列を使うのか（配列名や配列の大きさなど）をコンピュータに知らせなければなりません．これを配列の宣言といいます．

以下の各項で説明するように，宣言の仕方によって，いろいろな配列を作れます．

7·1·1 1次元の配列の宣言

配列を宣言するにはDIM文を使います．DIM文は，配列の名前とそこに記憶させるデータの型および配列の次元と大きさを指定する働きをします．

プログラムの中に書いてあるDIM文に出合うと，コンピュータはその指定に従って，

その配列の記憶場所を用意し，さらに，
その配列のすべての要素の値を，数値データの場合は0（ゼロ）に，
また文字データの場合はヌルストリング（長さ0の文字列）に設定します．

DIM文は次のような形に書きます．

DIM <配列名>(<大きさの指定>), ……

〔例 1〕 DIM A(10)

このDIM文は，Aという配列名で，10番までの配列要素を持った，1次元の配列を宣言しています．すなわち第7章の始めに図示した配列です．

大きさの指定はその配列の「要素番号の最大値」を書くのです．要素番号は必ず0（ゼロ）番から始まることにきまっています．したがってこの例では，0番から10番までの11個の要素が並んだ配列となります．

〔例 2〕 DIM A(10), B!(5), C%(100), D#(20), E$(12)

1つのDIM文で複数個の配列を宣言することができます．そのときはコンマ(,)で区切って並べます．この例では5個の配列を宣言しています．（これらを別々にしてDIM文をたくさん書いてもよろしい．）

配列名の付け方の規則は，普通の変数名の場合と同じです．

A(10)は型宣言文字が付いてないので，単精度型のデータを記憶する配列とみなされます．B!(5)も単精度型です．C%(100)は整数型，D#(20)は倍精度型，E$(12)は文字データの配列となります．

7·1·2　多次元の配列の宣言

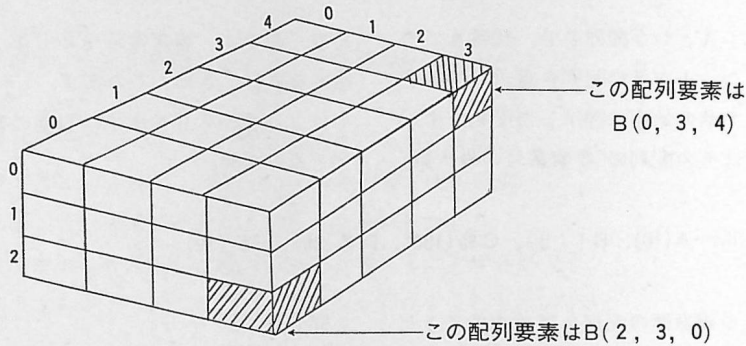
前項（7·1·1）で述べたのは1次元の配列ですが2次元，3次元，……などの多次元の配列も使うことができます．

1次元の場合と同様に，多次元の配列もDIM文で宣言します．ただし，大きさ指定が複数個必要です．すなわち，2次元ならば2個，3次元ならば3個の大きさ指定を，コンマ(,)で区切って並べてカッコの中に書きます．次の例を見てください．

〔例 1〕 DIM A(3, 4)

このDIM文は，Aという配列名で，2次元の単精度型の配列を宣言するものです．配列要素が並んでいる状態を，次のように図示するとわかり易くなります．

〔例 2〕 DIM B(2, 3, 4)

これは3次元の配列です．次のように図示することができます．（4次元以上の配列は図示することが困難ですが，必要に応じて使えば役立ちます．）

7·1·3 大きさを変えられる配列の宣言

前項までに示したDIM文の例では，配列の大きさの指定に定数を用いています．しかし，変数や式を使うことも許されています．ただし,そのDIM文より前の時点で，その変数や式に値を与えておくことが必要です．変数や式を使えば，配列の大きさをあらかじめ確定しないでおいて，プログラムを実行する段階で，そのつどきめることができます．

〔例 1〕
```
10 INPUT N
20 DIM A(N), B(2, N)
```

このDIM文では配列の大きさを指定するのに変数Nが使われています．10番の INPUT文によって，変数Nに5という値が入力されたとすれば，20番

DIM A(5), B(2, 5)

のDIM文は下記のDIM文と同じ働きをすることになります．

〔例 2〕
```
30 INPUT K, L
40 DIM C(K*L),D(K/L+1)
```

これは大きさ指定に式を用いた例です．

例えばKに20を，Lに3を入力すれば，40番の

DIM C(20* 3), D(20／3＋1)

DIM文は次のDIM文と同じ働きをすることになります．

20／3＋1＝7.66666ですが,このような場合には小数点以下は切捨てられて，結局は次の DIM

文と同じ働きをすることになります.

```
DIM  C(60), D(7)
```

配列の大きさを指定する変数や式の値がマイナスになった場合にはエラーとなります.

7・1・4　配列宣言の省略

配列を使う場合は，DIM文で宣言するのが原則です．しかし，もしDIM文を書かずに添字つき変数を使った場合は,「大きさ指定は10の配列が宣言されている」ものと自動的にみなして処理されることになっています.

いいかえれば，大きさが10以下の配列を使おうとする場合は，DIM文による宣言を省略してもよい，ということです.

〔例〕

```
DIM文なし
100 INPUT  A(J)
```

この100番のINPUT文を実行するときは,

```
DIM  A(10)
```

という配列宣言があるものとみなして処理されます.

したがって，もしJの値が10を越えればエラーとなります.

何かを省略できるということは便利ではありますが，その反面では，不注意によるプログラムミスの原因にもなり易いので，気を付けねばなりません.

7・2　配列宣言の取消し

DIM文によって宣言された配列は，それを取消さない限り，そのプログラムの中では存続して記憶場所を占有します．したがって，使い終って用済みになった配列はそれを取消して，空いた記憶場所を別の目的に利用することが考えられます．次に示すERASE文を使えば配列の取消しができます.

```
ERASE  <配列名>, <配列名>,……
```

配列名は，すでに宣言されている配列のうちで取消したいものの名前を，コンマ(,)で区切って並べて書きます.

〔例〕
```
10  INPUT  N
20  DIM  A(N), B(2, N)
        ∫
500 ERASE  A, B
510 GOTO  10
```

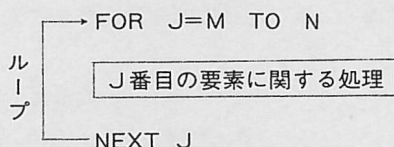
この例では，20番のDIM文で宣言された配列AとBは，500番のERASE文で取消されて，その記憶場所は解放されます．そこで再び20番のDIM文によって「新しい」配列AとBを宣言することができます．

すでに宣言されている配列を取消さずに，再びそれと同じ名前の配列を宣言すると，エラー(Redimensioned array)となります．

7·3　配列に対するくり返し処理

すでに述べたように，配列は多数の配列要素の集まりです．そして各配列要素は添字を使って呼ばれます．したがって配列を活用するには，添字をうまく使ってくり返し処理を行なうことが必要になります．その基本的な形を理解してもらうことが本節のねらいです．

一般にくり返し処理を行なうには，第4章(4·5)で紹介したFOR～NEXT文がよく使われます．配列処理の最も基本的なパターンは次のような形です．

```
    ┌→ FOR  J=M  TO  N
ル  │  ┌──────────────────┐
ー  │  │ J番目の要素に関する処理 │
プ  │  └──────────────────┘
    └── NEXT  J
```

すなわち，配列のM番目の要素からN番目の要素まで，順次に処理をくり返す形です．Mは1または0，Nはその配列の最大の要素番号とするのが普通の場合です．

もし配列が2次元のときには2重のループとなり，3次元ならば3重ループとなります．

以下に例を掲げて説明しましょう．

〔例1〕　1次元配列への入力,出力,クリアなど.
```
10  DIM  A(5)
20  FOR  J=0  TO  5  ←──┐
30  INPUT  A(J)          │ ループ
40  NEXT  J  ────────────┘
```

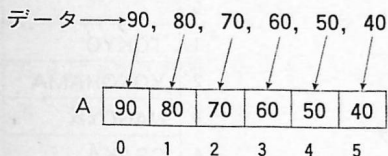
これは1次元配列Aの0番目から5番目までのすべての要素に，データを順次入力します．30番のステートメントが「J番目の要素に関する処理です．これを，

```
30  PRINT  A(J)
```

とすれば，画面に出力することになります．あるいは，

```
30  A(J)= 0
```

とすれば，すべての要素の値を0にすることができます．

もしこのプログラムをRUNして,次のデータをこの順に入力すれば，配列Aの中は図のようになります．

この例の処理を，FOR～NEXT文を使わずにひとつずつ書くと次のようになります．これを見てもFOR～NEXT文の効果がわかるでしょう．

```
10  DIM  A(5)
30  INPUT  A(0)
31  INPUT  A(1)
32  INPUT  A(2)
33  INPUT  A(3)
34  INPUT  A(4)
35  INPUT  A(5)
```

〔例2〕 配列要素の値を合計する．

```
50  S=0
60  FOR J=0  TO 5 ←――┐
70  S=S+A(J)          │ ループ
80  NEXT J  ――――――――┘
90  PRINT "ゴウケイ=";S
```

これは現列Aの0番目から5番目までのすべてのデータを順次Sに加算します．これと上の〔例1〕とを連結してRUNすれば，画面には次のように出力されます．

```
ゴウケイ=390
```

〔注〕390=90+80+70+60+50+40

ところで,〔例1〕と〔例2〕を連結するのに，次のようにまとめることもできます．

```
10  DIM  A(5):S=0
20  FOR  J=0   TQ  5 ←┐
30  INPUT  A(J):S=S+A(J)   │ループ
40  NEXT J ─────────┘
50  PRINT "ゴウケイ="; S
```

〔例 3〕 配列へDATAをREADする.

```
10  DIM  A$ (6)
20  FOR  J=1   TO 6 ←┐
30  READ  A$(J)       │ループ
40  NEXT  J ────────┘
100 DATA  TOKYO, YOKOHAMA, NAGQYA
101 DATA  OSAKA, HUKUOKA, SAPPORO
```

これは，文字型配列A＄の中へDATA文のデータを入れています．その結果，配列の中は図のようになります．

この例ではA＄(0)という要素は使っていません．

〔例 4〕 2次元配列の場合.

```
10  DIM  A(3, 4)
20  FQR  I=1   TO  3 ←──────┐
30   FOR  J=1   TO  4 ←┐      │
40   INPUT  A(I, J)     │Jループ │Iループ
50   NEXT  J ─────────┘      │
60  NEXT  I ──────────────┘
```

これは2次元配列　にデータを入力します.(0行目，0列目の場所は使っていません.)ループを2重にしてあります，考え方は次のとおりです.

外側のＩのループは，1行分ずつ，1行目から3行目までくり返す.
内側のＪループは，1行分の中において，1個ずつ，1列目から4列目までくり返す.

もしRUNして，次のデータをこの順に入力すれば，配列の中は右図のようになります.

データ→1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

7・4　配列データの並べ替え

いろいろな問題をコンピュータで処理する場合，でたらめな順序に並んでいるデータを「ある一定の順序に並べ替える」ことがしばしば必要になります．

数値データならば大きさの順（小→大，あるいは大→小）に並べ替える，また文字データならばアルファベット順（A，B，C，……）に並べ替える，などです．

このような並べ替えのことをコンピュータの世界では「分類」とか「ソート(sort)」と呼びます．本節では，配列内のデータを並べ替える方法を紹介します．

7・4・1　数値データの並べ替え

次に示す例は，10個の数値データを「小さい」ものから順に並べ替えるものです．もし逆に「大きい」ものから順に並べる場合は，70番のIF文の中の不等号を逆にして

```
70 IF A(J)<=A(I) GO TO 90
```

とすればよいのです．また，データが10個より多くても少なくても，要領は同じです．要するに並べ替えの考え方を理解して応用することが大切です．

〔例〕　10個のデータの入力，並べ替え，出力

```
10  DIM A(10)          ←配列の宣言
20  FOR J=1 TO 10      ┐
30  INPUT A(J)         │入力（10個のデータ）
40  NEXT J             ┘
50  FOR I=1 TO 9       ┐
60  FOR J=I+1 TO 10    │
70  IF A(J)>=A(I) GOTO 90 │大きさを比較して，並べ替える．
80  SWAP A(J), A(I)    │
90  NEXT J             │
100 NEXT I             ┘
110 FOR J=1 TO 10      ┐
120 PRINT J, A(J)      │出力（10個のデータ）
130 NEXT J             ┘
140 END
```

この例の20〜40番のステートメントはデータを力する部分です，また110〜130番のステートメントは，並べ替えられたデータを出力する部分です．これらについては7.3〔例1〕で説明しまし

た. したがって, 並べ替えの作業をする部分は50～100番のステートメントでは,

その考え方の要点は次のとおりです.

① 配列内のデータを2個取出して,大きさを比較する.

② もし大きさが逆順であれば, 配列内の位置を交換する. (逆順でなければ, 交換せずそのままとする.)

③ 上記のことを順にくり返す.

70番のIF文で, J番目のデータとI番目のデータとを比較しています.
80番のSWAP文で2個のデータを交換しています.

ここでSWAP文の書き方を紹介しておきましょう.

```
SWAP <変数名>, <変数名>
```

SWAP文の機能は, 2つの変数値を交換することです. ただし, 2つの変数の型(整数型, 単精度型, 倍精度型, 文字型)は同じでなければいけません. 違った型の変数に対してSWAPをさせようとすると, エラー(Type mismatch)となります.

80 SWAP A(J), A(I) というステートメントによって, 配列の中の状態は次のようになります.

A(J)の値	A(I)の値	
23	45	……SWAP文 実行前の状態
↓	↓	
45	23	……SWAP文 実行後の状態

配列データの並べ替えのときに限らず, 必要に応じてSWAP文を利用してください.

7·4·2 文字データの並べ替え

ここでは例として, 10個の文字データを配列内に入力し, 各データの最初の1文字に注目して, アルファベット(ABC)順に並べ替えて出力するプログラムを示します. 各データの最初の1文字は英大文字とします. そうでないデータが入力された場合は, 入力のやりなおしをさせるようにします.

並べ替えの考え方は前項の例と同様ですが, データの比較は文字コードによって行ないます. 文字コードについては第6章を参照してください.

文字Aの文字コードは65, Bの文字コードは66, Cは67, Zの文字コードは90です. したがって文字コードを比較して, その小さい順に並べ替えればよいことになります.

〔例〕

```
10  DIM  A$(10)
20  FOR  J=1   TO  10
30  INPUT  A$(J) : X= ASC(A$(J))
40  IF  X<65   OR  X>90   THEN
    PRINT "INPUT  AGAIN!" : GOTO  30
50  NEXT  J
100 FOR  I=1   TO  9
110 FOR  J=I+1   TO  10
120 IF  ASC(A$(J))<ASC(A$(I))
     THEN  SWAP   A$(J), A$(I)
130 NEXT  J
140 NEXT  I
200 FOR  J=1   TO  10
210 PRINT  J, A$(J)
220 NEXT  J
300 END
```

（10〜50：入力，100〜140：並べ替え，200〜300：出力）

理解度テスト

(1) 次の文のうち，正しいものには○をつけ，誤りのものには×をつけなさい．

① 配列の要素番号は，必ず１番から始まる．
② 文字型データの配列では，１個の配列要素に１文字ずつ記憶させる方式になっている．
③ ひとつのDIM文によって，複数個の配列を宣言することができる．
④ 宣言された配列は，プログラムの実行が終了するまでは，取消すことができない．
⑤ひとつのDIM文の中で，１次元配列と２次元配列とを同時に宣言してもよい．
⑥ 配列の添字の値は，０（ゼロ）になってはいけない．
⑦ ひとつのDIM文の中で，数値型配列と文字型配列とを同時に宣言してはいけない．
⑧ 配列の添字の値は，マイナスになってはいけない．
⑨ ひとつの配列の中に，数値型データと文字型データとを混合で記憶させることはできない．
⑩ ERASE文は，配列内のデータの値を０（ゼロ）にするためのステートメントである．

(2) 次のプログラムによって宣言される配列A，B，C＄は，それぞれ何個の要素を持っているか，答えなさい．

```
10  INPUT  N    ←――― Nに5を入力する
20  DIM  A(N+1)
30  DIM  B(N, 4)
40  DIM  C$(2, 3, N)
```

(3)　5行×5列の大きさの2次元配列を宣言して，その配列の中に次図のようにデータを記憶させるプログラムを作りなさい．ただし，データの入力は行なわないものとします．

①

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

②

1	2	3	4	5
2	3	4	5	6
3	4	5	6	7
4	5	6	7	8
5	6	7	8	9

③

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

(4)　次図のように，平年の各月の日数を配列内のA(1)からA(12)までの要素に，DATA文から読み込むプログラムを作りなさい．

10　DIM　A(12)

0	31	28	31	30	31	30	31	31	30	31	30	31
0	1	2	3	4	5	6	7	8	9	10	11	12

第8章　画面への出力

本章では，画面への出力の方法について解説します．画面への出力とは，PC－8001に接続されたディスプレイ装置の画面に，数値や文字を表示すること，あるいはドット（点）によって図形を

表示することです.

使用するディスプレイ装置は，次の2つのうちのいずれかです.

① 12インチ・グリーンディスプレイ(PC－8041)

② 12インチ・カラーディスプレイ（PC－8043)

上記①はいわゆる白黒テレビと同じで，カラー（色）を出すことはできません．②はカラーテレビに相当するもので，8種類の色を使って表示することができます.

また，表示の形態には次の2つがあります.

① キャラクタ表示：

これは，文字を使って表示する形です．文字とは，数字・英字・カナ文字・図形文字など，キーボードの普通キーに付いている各種の文字です．これらの文字を画面に並べて表示することにより，数値データや文字データの内容を眼で見ることができるわけです．コンピュータによる計算処理の結果を出力するために，もっとも普通に用いられる方法です．キャラクタ表示に関するいろいろな命令や指定の仕方がありますが，もっとも基本となるのはPRINT文というステートメントです.

② グラフィック表示：

これは，ドット（点）を使って表示する形です．ドットとは，キャラクタ表示における1文字分の画面領域をさらに8等分した小さな点です．この点を画面に並べて表示することにより，グラフや絵などの図形を書くことができるわけです．キャラクタ表示でも，文字を並べて図形を書くことはある程度できますが，どうしても粗い表現になります．グラフィック表示を用いれば，かなり細密な表現が可能となります．グラフィック表示に関するいろいろな命令や指定の仕方がありますが，その基礎となるのはPSET文です.

上述のことをまとめると，次表のように4種の表示状態があることになります.

	キャラクタ表示	グラフィック表示
色なし(白　黒)	白黒， キャラクタ表示	白黒， グラフィック表示
色つき(カラー)	カラー， キャラクタ表示	カラー， グラフィック表示

この4種の表示状態のうちのどれかひとつを指定して画面表示を行なうわけですが，場合によってはこれらを混用することもできます．もっとも簡単なのは（白黒，キャラクタ表示）です．もっとも複雑なのは（カラー，キャラクタ表示とグラフィック表示の混用）です．表示状態が複雑な場合は，ある種の制限条件が適用されます.

画面への出力については，以上の他にいろいろなことを指定する必要があります．それらについて，以下の各節で解説します.

8·1　画面の表示状態の指定

画面への出力に際しては，ただ単にPRINT文やPSET文を書くだけでなく，表示状態に関する各種の条件を指定しなければなりません．画面全体に対する条件設定もあれば，部分的な表示状態の指定もあります．このような指定を行なうために使われる次の4つのステートメントについて，本節で解説します.

WIDTH文，CONSOLE文，COLOR文，LINE文

それぞれのステートメントの説明に入る前に，

それらが何を指定するものなのかを，表にまとめて示しておきます．

指定対象	指定事項 ＼ ステートメントおよび使用時のモード	WIDTH文	CONSOLE文	COLOR文		LINE文
				白黒モード	カラーモード	カラーモード
画面全体	表示する桁数，行数の指定	○				
	スクロール範囲の指定		○			
	fキーの表示の有無の指定		○			
	白黒/カラーモードの指定		○			
	グラフィックスイッチの指定			○	○	
	ヌルキャラクタコードの指定			○	○	
キャラクタ単位	色の種類の指定				○	
	ブリンク，リバースの指定			○		
行単位	ブリンク，リバースの指定					○

8·1·1　WIDTH文

WIDTH文は，画面に表示する文字の桁数と行数とを指定するために使うステートメントです．

```
WIDTH　〈桁数〉,　〈行数〉
```

桁数とは画面の1行あたりに表示する文字数であり，36, 40, 72, 80のうちのいずれかを指定します．36あるいは40の場合は，文字は普通の大きさで表示されます．したがって，数値データや文字データを普通の形でキャラクタ表示する場合や，プログラムを入力する場合などに適しています．72あるいは80を指定すると，文字の横幅が，36あるいは40の場合の半分になります．つまり文字の形が縦長になります．したがって，文字としてはやや見にくくなりますが，細密なグラフィック表示には適しています．

なお，カラーモードの場合には，画面に実際に表示できる桁数は，WIDTH文で指定した桁数より1桁少なくなります．例えば，80と指定しても，実際に表示できるのは79桁です．(カラーモードについては次項を参照)

次に桁数ですが，これは20または25のいずれかを指定します．20の場合は行と行との間に細いすき間ができます．キャラクタ表示の場合はその方が見やすいのですが，グラフィック表示の場合は，縦の線を書くと4ドットごとにすき間ができます．25の場合はこのすき間はできません．したがってグラフィック表示をするには25行の方が適しているといえましょう．

なお，電源投入時には，画面の状態は自動的に40桁，20行にセットされます．またWIDTH文は，プログラムの中でも，あるいは直接モードでも使用できます．

〔例1〕 WIDTH　80, 25

このWIDTH文は，画面に表示する文字数を80桁×25行（＝2000文字）に指定するものです．

〔例2〕 WIDTH 40 ←― 行数指定の省略

このWIDTH文には行数の指定が書いてありませんが，これは指定を省略したのです．WIDTH文を実行する直前の状態に対して，行数は変化させずに桁数のみを変える場合は，行数の指定を省略してもよいのです．この例のWIDTH文を実行する直前の状態が，もし80桁×25行になっていたとすれば，このWIDTH文を実行することによって，画面は40桁×25行の状態となります．すなわち，省略せずに次のように書いたのと同じ結果になります．

WIDTH 40, 25

〔例3〕 WIDTH , 20 ←― 桁数指定の省略

この例では桁数の指定が書いてありません．桁数指定を省略するときは，コンマ（,）を書かなければいけません．〔例2〕のように行数指定を省略するときは，コンマは不要です．

桁数指定を省略すると，桁数は直前の状態のままで変化しません．この例のWIDTH文を実行する直前の状態が，もし40桁×25行になっていたとすれば，このWIDTH文を実行することによって，画面は40桁×20行の状態となります．すなわち，省略せずに次のように書いたのと同じ結果になります．

WIDTH 40, 20

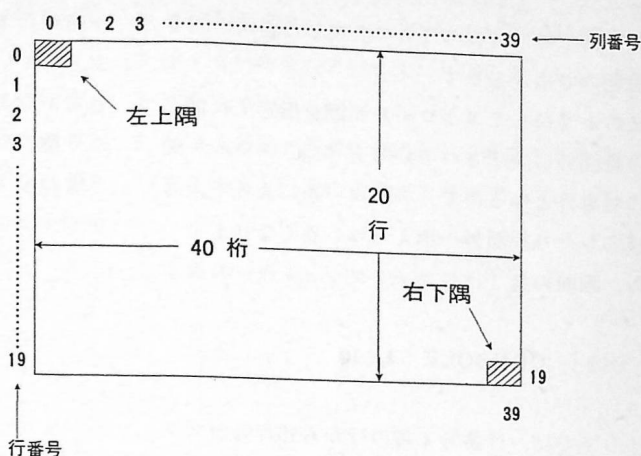
WIDTH文の書き方は以上に述べたとおりであり，桁数と行数の指定の仕方の組合わせは次のうちのどれかになります．最大は80桁×25行（＝2000文字），最小は36桁×20行（＝720文字）です．

	36桁	40桁	72桁	80桁
20行	36, 20	40, 20	72, 20	80, 20
25行	36, 25	40, 25	72, 25	80, 25

このように指定された桁数・行数の範囲内で画面表示が行なわれるわけですが，この各桁・各行には番号がつけられています．これを，画面の「列番号」・「行番号」と呼ぶことにします．この番号によって，画面内の任意の（1文字分の）位置を表わすことができるわけです．画面表示に関するいろいろな命令を書くときに，この番号が使われます．

列番号および行番号は，必ず0番から始まります．WIDTH文で指定された画面の表示範囲において，左上隅の位置の番号が0番です．列番号は，0番から右に向って順に1番・2番・3番………とつけられます．行番号は，0番から下に向って順に1番・2番……とつけられます．左上隅は1番ではなく0番だということを忘れないように注意してください．

列番号・行番号のつけ方を次図に示しておきます．これは下記のWIDTH文で指定された画面です．

WIDTH　40, 20

8·1·2　CONSOLE 文

CONSOLE 文は次の3つのことを指定するために使うステートメントです．

① スクロール範囲（スクロール開始行とスクロール行数）.

② ファンクションキーの表示の有無.

③ 白黒モードかカラーモードか.

CONSOLE 文は次のような形に書きます.

CONSOLE〈①の開始行〉,〈①の行数〉,〈②の指定〉,〈③の指定〉

〔例 1〕 CONSOLE　0, 15, 0, 1

この例では，スクロール範囲は画面の最上行から15行分であり，ファンクションキーの内容は画面に表示せず，そしてカラーモードにする，ということを指定しています.

それぞれの指定の書き方やその意味について，以下に説明を加えます.

なお，CONSOLE 文に書くそれぞれの指定内容がその前の状態と変わらないときは，WIDTH 文の場合と同じ要領で，指定を省略することができます.

〔例 2〕 CONSOLE　, , 0, 1

この例では，スクロール範囲の指定（開始行と行数）を省略しています.

(1)　スクロール範囲

キーボードのキーを押したり，あるいはPRINT文を使ったりして画面に文字を書く場合，画面の最下行まで降りてきて，さらにその下に書こうとすると，画面に表示されていた文字全体が1行分だけ上に移動して，最上行にあった文字は画面から消えます。このような画面の動きを「スクロール」といいます.

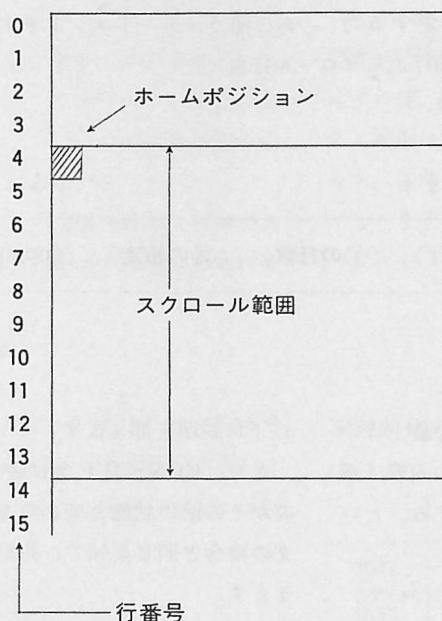
このスクロールの動作範囲を CONSOLE 文によって指定することができます．そのために，スクロール開始行および行数を書くわけです．スクロール開始行を指定するには，前項（8·1·1）で述べた行番号を用います．したがって，0を指定すれば最上行がスクロール開始行となります．スク

ロール行数は，WIDTH文で指定した画面の広さの範囲内で指定します．

このようにしてスクロール範囲を指定すれば，その範囲外に表示されている文字はスクロール動作の対象外となるので，スクロールによって上方へ移動したり画面外へ消えたりしなくなります．なお，画面の最下行にファンクションキーの表示があるときは，この行は（スクロール範囲内に指定されていても）スクロール動作の対象になりません．なお，スクロール範囲のことをスクロールの「窓」(ウィンドウ）とも呼びます．

電源投入時には自動的に，スクロール開始行＝0，スクロール行数＝20にセットされます．

〔例3〕 CONSOLE 4，10

この例では，行番号4番の行から10行分がスクロール範囲となります．次図を見てください．なお，スクロール範囲以外の指定は省略されています．

(2) ファンクションキーの表示の有無

この指定を1にすると，画面の最下行にファンクションキーの定義内容が表示されます．指定を0にすると表示されません．電源投入時には自動的に1がセットされます．

(3) 白黒モードとカラーモード

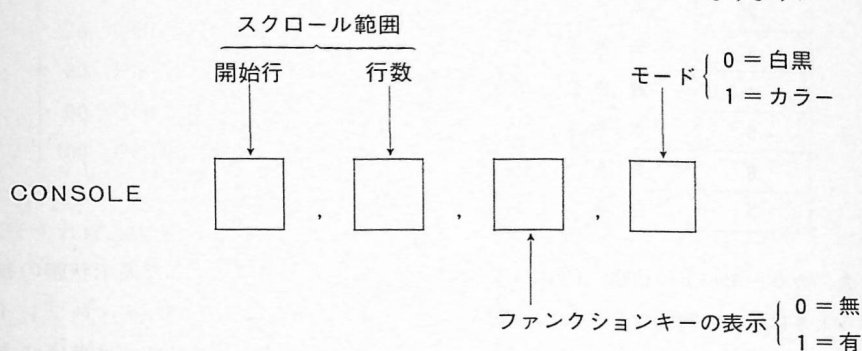
この指定を0にすれば白黒モードとなり，1にすればカラーモードとなります．

白黒モードとかカラーモードというのは，画面の性質（表示動作の様式）であり，いずれか一方を選んで指定して用います．この白黒／カラーモードは，ディスプレイ装置の種類（単色ディスプレイ装置かカラーディスプレイ装置か）とは別のものです．カラーディスプレイ装置を使って，色つきで画面表示をするときは，カラーモードにしなければ不可能です．しかし，カラー装置を白黒モード指定で使うこともできます.（ただし，色はつきません.）また，単色装置をカラーモード指定で使うこともできます.（ただし，色はつきません.）

通常は，カラー装置で色つき表示をする場合は

カラーモード指定とし，その他の場合は白黒モード指定とします．

同じ命令でも，それが実行されるときの状態（白黒モードかカラーモードか）によって，働きが異なる場合があります．次項（8·1·3）のCOLOR文がそうです．また，カラーモードの状態でしか使えない命令もあります．LINE文（8·1·4参照）がそうです．

以上に述べたCONSOLE文の書き方の要点をまとめると，次のようになります．

8·1·3　COLOR文

COLOR文は次の3つのことを指定するために使うステートメントです．

① 機能コード（色の種類，または，ブリンク・リバースなど）

② ヌルキャラクタコード

③ グラフィックスイッチ

COLOR文は次のような形に書きます．

COLOR 〈①の指定〉, 〈②の指定〉, 〈③の指定〉

〔例1〕 COLOR 2, 0, 1

この例では，機能コードは2を指定し，ヌルキャラクタコードは0（空白），グラフィックスイッチは1（グラフィックモード）を指定しています．

それぞれの指定の書き方やその意味について，以下に説明を加えます．

〔例2〕 COLOR ，，1

この例では，機能コードおよびヌルキャラクタコードの指定を省略しています．

なお，COLOR文に書くそれぞれの指定内容がその前の状態と変わらないときは，WIDTH文やCONSOLE文の場合と同じ要領で，指定を省略することができます．

(1) 機能コード

機能コードは，0～7の範囲の数字で指定します．その意味は，画面が白黒モードであるかカラーモードであるかによって異なります．それを次

表に示します．

機能コード	カラーモードのとき	白黒モードのとき
0	黒　色	ノーマル
1	青　色	シークレット
2	赤　色	ブリンク
3	紫　色	シークレット
4	緑　色	リバース
5	水　色	リバース＋シークレット
6	黄　色	リバース＋ブリンク
7	白　色	リバース＋シークレット

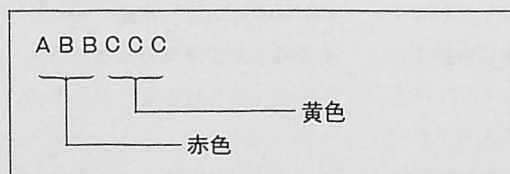
すなわち，カラーモードの状態でCOLOR文が実行されるときは，その機能コードは色の種類を指定する働きをします．そして，カラーディスプレイ装置であれば，その後の文字は指定された色で表示されます．もし単色ディスプレイ装置であれば，色はつきませんが，指定された色の種類に応じて文字の表示の濃淡（輝度）が変わります．

また，白黒モードの状態でCOLOR文が実行されるときは，その機能コードは，色の種類ではなく，上表に掲げたような表示状態の種類を指定する働きをします．カラーディスプレイ装置であっても，白黒モードの状態では色はつきません．

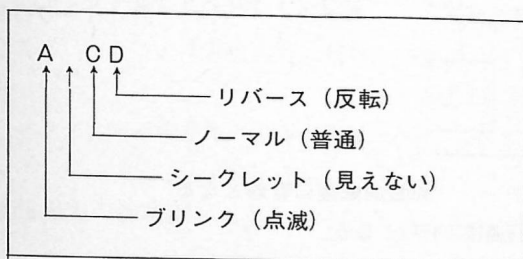
上表の中の「ノーマル」というのは，普通の表示状態（電源投入時と同じ）です．「シークレット」を指定すると何も表示されません．「ブリンク」を指定すると文字が点滅します．「リバース」は反転です．反転とは写真のネガとポジの関係のようなものです．

〔例3〕

```
20  CONSOLE  , , , 1      ←―――― カラーモードを指定
30  COLOR  2, 0, 0       ←―――― 赤色を指定
40  PRINT  "A";
50  PRINT  "BB";
60  COLOR  6             ←―――― 黄色を指定
70  PRINT  "CCC"
```

カラーディスプレイ装置を使って実行すれば，画面は次のようになります．

〔例4〕

```
 20  CONSOLE  , , , 0     ←―――― 白黒モードを指定
 30  COLOR  2             ←―――― ブリンクを指定
 40  PRINT  "A";
 50  COLOR  1             ←―――― シークレットを指定
 60  PRINT  "B";
 70  COLOR  0             ←―――― ノーマルを指定
 80  PRINT  "C";
 90  COLOR  4             ←―――― リバースを指定
100  PRINT  "D"
```

これを実行すれば，画面は次のようになります．

(2) ヌルキャラクタコード

ヌルキャラクタというのは，画面をクリア（消去）したときに画面全体に表示される文字のことです．つまり画面の背景（バック）です．普通は0を指定します．これは空白です．

ヌルキャラクタとして空白以外の文字を指定することもできます．それには，その文字の文字コードを使います．文字コードについては第6章(6·1)を参照してください．

〔例5〕

```
20  CONSOLE  , , , 0     ←―――― 白黒モードを指定
30  COLOR  0, 65, 0      ←―――― 65は文字Aの文字コード
40  PRINT  CHR$ (12)     ←―――― 画面をクリア
```

この例では，ヌルキャラクタとして「A」という文字を指定しています．Aの文字コードは65（10進法）です．これは41（16進）ですから，16進表示で &H41 と書いてもよろしい．

指定したヌルキャラクタは，COLOR文を実行しただけでは画面には現われません．その後に画面消去をしたときから画面に現われます．この例では，40番のPRINT文を実行した時点で，画面全体がAという文字で埋められます．

(3) グラフィックスイッチ

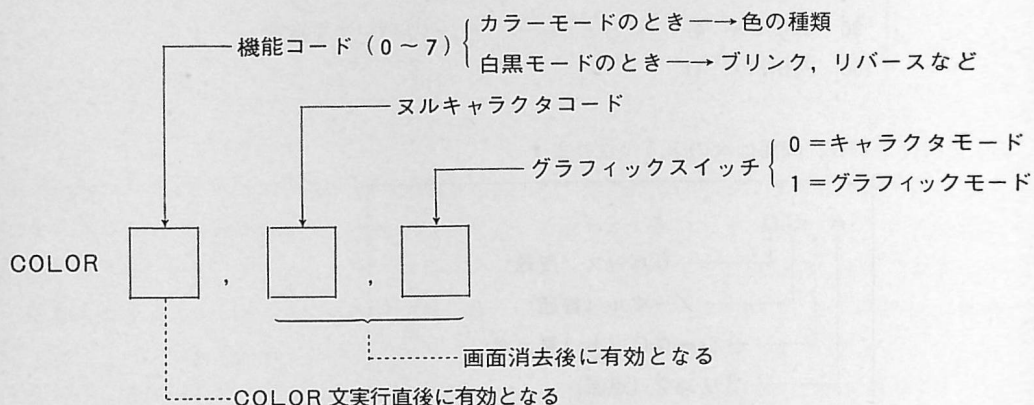
この指定を0にすればキャラクタモードとなり，1にすればグラフィックモードとなります．

キャラクタ表示（文字による表示）のみを行なうときはキャラクタモードを指定し，また，グラフィック表示（ドットによる表示）のみを行なうときはグラフィックモードを指定します．2種の表示を混用することもできますが，その場合は多く使う方の表示に合わせて，グラフィックスイッチを指定した方がよろしい．

〔例6〕

```
30 COLOR , , 1          ←―――― グラフィックスイッチを1に指定
40 PRINT  CHR$ (12)     ←―――― 画面をクリア
```

指定したグラフィックスイッチは，COLOR文を実行しただけではその効力を現わしません．その後に画面消去をしたときから有効となります．この点はヌルキャラクタコードの指定の場合と同様です．

以上に述べたCOLOR文の書き方の要点をまとめると，次のようになります．

8·1·4　LINE文

このLINE文は，カラーモードの状態においてのみ使用できます．白黒モードでは使えません．

1行単位で，表示の仕方（ブリンク，リバースなど）を指定するためのステートメントです．

```
LINE 〈行番号〉,〈機能コード〉
```

行番号は画面上の行の位置を指定するものであり，8·1·1で説明した番号を用います．すなわち，最上行の行番号は0番です．

機能コードは0～3の範囲の数字で指定します．その意味は次表のとおりです．

機能コード	意　味
0	ノーマル
1	ブリンク
2	リバース
3	リバース＋ブリンク

前項で述べたように，白黒モードの状態では，COLOR文によってブリンク，リバースなどを1桁ごとに指定することができます．しかしカラーモードの状態では，ブリンク，リバースなどの指定はCOLOR文ではできないので，LINE文を使うことになります．

LINE文は，画面上の指定された1行に対して，行単位で動作します．すなわち指定された1行の

中のすべての桁がその働きを受けます．１桁ごとの指定はできません．

〔例〕

```
10  CONSOLE  ,,0,1   ←── カラーモードを指定
   ⸾
100 LINE 2,1         ←── ブリンクを指定  ┐
   ⸾                                      ├ この間はブリンクが続く
200 LINE 2,0         ←── ノーマルを指定  ┘
```

8·2　数値・文字の出力

画面への出力形式として通常よく用いられるキャラクタ表示の方法について，本節と次節で解説します．

本節では，キャラクタ表示の基本となるPRINT文およびそれに関連する表示位置の指定方法などについて説明します．次節では，PRINT USING文による編集の方法を紹介します．

8·2·1　画面出力の基本形

数値データや文字データを画面に出力する際の基本的なステートメントはPRINT文です．また，機能文字を使うことによって，画面消去その他の働きをさせるためにもPRINT文が用いられます．

PRINT文は次のような形に書きます．

PRINT〈出力並び〉　または　?〈出力並び〉

疑問符（?）は，PRINTという語の省略形です．

出力並びというのは，出力したい項目を，コンマ（,）またはセミコロン（;）で区切って並べたものです．出力並びに並べる項目としては，変数名，定数，あるいは式を書くことができます．それらは数値型でも文字型でもよろしい．ただし，文字定数は引用符(″)で囲まなくてはいけません．

PRINT文を実行すると，出力並びの各項目の値が，順に画面に表示されます．もし出力並びを全然書かないと，単に改行だけが行なわれます．

出力並びの各項目が表示される位置や項目間の間隔は，区切りとしてコンマを用いるかセミコロンを用いるかによって異なります．またその項目が数値データか文字データかによっても違います．それらは，以下に述べるような規則に従って行なわれます．

出力項目をコンマで区切った場合は，そのコンマの直後に書かれている項目の表示開始位置は，下記のような標準化された位置となります．(14桁ごとの位置)

WIDTH文の桁数指定	表示開始位置の列番号
40桁の場合	0，14
80桁の場合	0，14，28，42，56

セミコロンで区切った場合は，そのセミコロンの直後に書かれている項目の表示開始位置は，その前の項目の直後となります.（上記の標準位置とは無関係です.）

出力並びの末尾にコンマもセミコロンも無い場合は，最後の項目を表示し終わると自動的に改行されます.

出力並びの末尾にコンマまたはセミコロンがある場合は，最後の項目を表示し終わっても改行せず，上記の標準位置または直後の位置が，次のPRINT文による表示開始位置となります.

なお，画面の行の右端までに表示しきれない場合は，自動的に改行して表示します.

また，数値を表示すると，その直後に空白が1桁置かれます．また数値表示の開始位置は符号桁として用いられます．数値が負の場合はマイナス記号（－）が符号桁に表示され，数値が正の場合は符号桁は空白となります.

数値の絶対値が非常に大きい場合や非常に小さい場合は，指数形式で表示されます．例えば，単精度型のデータで値が0.00000001のときは，1E－8 と表示されます.

整数部分が0（ゼロ）の小数(例えば，－0.123)は，整数部分の0は表示されません.（すなわち，－.123と表示されます.）

〔例1〕

```
50 X＝123 : N$＝"ABC"
60 PRINT X, X－200           ←── コンマで区切る
70 PRINT －X; X－200;        ←── セミコロンで区切り,末尾にセミコロンをつける
80 ? N$; "DEF"; 12.3         ←── 省略形（?）を使用
```

これを実行すると，画面には次のように表示されます.

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 ←── 列番号

  1 2 3                       －7 7
－1 2 3   －7 7   A B C D E F    1 2 . 3
```

〔例2〕

```
10 FOR J＝1 TO 5
20 PRINT J;
30 NEXT J
```

これを実行すると，画面には次のように表示されます.

〔例3〕

```
10 INPUT X, Y
20 PRINT "X＝"; X; "Y＝"; Y
30 PRINT "X＝" X "Y＝" Y
```

これを実行すると，画面には次のように表示されます．20番のPRINT文と30番のPRINT 文とは，同じ形に出力します．引用符（″）とセミコロン（；）とが隣り合わせになっている場合は，この例のようにセミコロンを省略することができます．

```
 0 1 2 3 4 5 6 7 8 9 10 11 12   ←── 列番号
┌──────────────────────────────
│ ?   1 2 3 ,  4 5 6 ←──── INPUT
│ X=    1 2 3   Y =   4 5 6
│ X=    1 2 3   Y =   4 5 6
```

〔例4〕　100　PRINT　CHR$（12）

これを実行すると，画面がクリア（消去）されます．そして，カーソル（表示位置）はホームポジションへ移動します．12は「画面消去」という働きを意味する機能文字の文字コードです．CHR＄関数については第6章を参照してください．

PRINT文の中にCHR＄関数を使うことによって，各種の画面操作を行なうことができます．下記のようなものがあります．

CHR$（12）	画面消去して，カーソルをホームポジションへ移す．
CHR$（11）	カーソルをホームポジションへ移す．（画面消去はしない）
CHR$（10）	改行する．
CHR$（13）	カソールを行の左端に移す．　カーソル
CHR$（9）	カーソルを8桁ずつ右へ移す．
CHR$（28）	右へ ┐
CHR$（29）	左へ │ カーソルを1文字分だけ動かす．
CHR$（30）	上へ │
CHR$（31）	下へ ┘
CHR$（7）	ブザーを1回鳴らす．

8·2·2　　表示位置の指定

本項では，画面に文字を表示するときにその表示位置を指定する方法を説明します．

PRINT文において，区切り記号（コンマやセミコロン）を使い分けることによって，あるいはCHR＄関数を使ってカーソルを移動させることによって，表示位置をある程度コントロールできます．しかしそれだけではまだ不便です．そこで，以下に紹介するような方法を利用します．

(1)　カーソル位置の指定

画面上の任意の位置にカーソルを移動させるために，次のLOCATE文を使います．

```
LOCATE 〈列番号〉,〈行番号〉,〈カーソルスイッチ〉
```

列番号と行番号は，画面上の位置を指定するものであり，画面の左上隅の位置が0番です.（8·1·1参照）

LOCATE文を実行すると，カーソルは指定された位置に移動します．移動するといっても，プログラム実行中は（INPUT文などで入力待ちのときを除き）カーソル（■）は画面には現われず，目には見えません．しかしコンピュータはいつでもカーソルの現在位置を記憶しているのです．カーソルは次に表示すべき画面上の位置を指すものです．

したがって，LOCATE文を使ってカーソルを移動してからPRINT文を実行させると，LOCATE文で指定した位置（移動後のカーソル位置）が，PRINT文による表示開始位置となるわけです．

LOCATE文中のカーソルスイッチとしては，1または0を書きます．

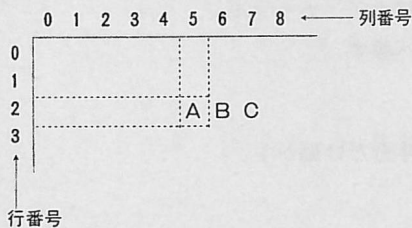
1を指定すると，カーソルが画面に表示されます．ただし，上述のように，プログラム実行中は表示されません．電源投入時には，カーソルスイッチは自動的に1がセットされます．

0を指定すると，カーソルは画面に表示されなくなります．プログラム実行中に限らず，コマンドレベルの状態においても表示されません．

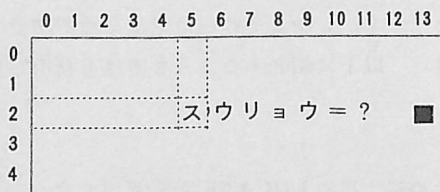
なお，LOCATE文のカーソルスイッチは省略することができます．省略すると，それ以前の状態が変わらずに続きます．

〔例1〕

```
10 LOCATE 5 , 2          ←――― カーソルスイッチの指定を省略
20 PRINT "ABC"
```

これを実行すると，画面に次のように表示されます．

〔例2〕

```
50 LOCATE 5, 2, 1
60 INPUT "スウリョウ="; S
```

この例ではINPUT文と組合わせてLOCATE文を使っています．このようにすれば，画面位置を指定して入力を行なうことができます．画面は次のようになります．

(2) カーソルの現在位置

画面上の表示が進行するにともなって，カーソル位置も動いて行きます．そのカーソルの現在位置を知りたい場合には，次の２つの関数を利用します．

POS（J）	………カーソルの現在位置の列番号を求める

カッコの中の引数（J）はダミー（形式的なもの）であり，その値は任意です．

CSRLIN	………カーソルの現在位置の行番号を求める

〔例３〕

```
10 LOCATE  5, 2
20 PRINT  POS (0) ; CSRLIN
```

これを実行すると，画面には図のように表示されます．

〔例４〕

```
50 LOCATE  POS (J) +R, CSRLIN +D
60 PRINT  X
```

60番のPRINT文のXの表示開始位置は，カーソルの現在位置よりR桁右で，D行下の位置となります．

Rの値が負ならばR桁左となり，Dの値が負ならばD行上となります．このようにすれば，カーソルの現在位置を基点として，画面上の表示位置を指定することができるわけです．

(3)　桁位置の指定

ひとつのPRINT文の中で，表示の桁位置を指定するために，次のTABとSPCを利用できます．この２つのものは，PRINT文の中またはLPRINT文（第９章参照）の中でのみ使うことができます．

TAB（J）	………第Ｊ番目の桁までカーソルを右へ移動する
SPC（J）	………Ｊ個の空白を表示する

TABでもSPCでも，Ｊの値は０～255の範囲内でなければなりません．

TABにおいては，カーソルの現在の桁の番号がＪの値を超えていれば，そのTABは無効となります．桁の番号は，PRINT文による表示が開始される行の左端の桁を０番として数えます．

〔例5〕

```
10 PRINT TAB (2) "ABC" TAB (6) "DEF"
20 PRINT SPC (2) "ABC" SPC (6) "DEF"
```

これを実行すると，画面には図のように表示されます．TABとSPCの違いに注意してください．

8·3 数値・文字の編集

前節で述べた方法を使えば，数値や文字を画面に出力できるわけですが，本節で述べるPRINT USING文を使えば，さらにいろいろな編集を加えて，画面表示をより良いものにすることができます．

編集のための各種の記号があるので，以下の説明を読んでその使い方を理解して利用してください．

数値データや文字データを編集して画面へ出力するためにPRINT USING 文を使います．

```
PRINT USING "〈書式仕様〉"；〈出力並び〉
```

書式仕様は，右に掲げる各種の書式記号を使って書きます．また，書式記号以外の文字を書くこともでき，それらの文字（空白も含む）はそのまま画面に出力されます．

出力並びには，数値型あるいは文字型の変数名，定数，式などを書きます．

書式記号は，出力並びの中の対応するデータを画面に出力するときの，画面の幅（桁数）および編集形式を指定する働きをします．各種の書式記号のあらましを次表に示します．

この表において，下の2つは文字データ用であり，他のものはすべて数値データ用です．数値データ用の書式記号の中では＃が基本的なものであり，その他の記号は＃と組合わせて用います．

数値データの編集に際しては，表示される数字の右端桁は，その下の桁を4捨5入して丸めたも

書式記号	摘　　要
＃	数字を表示する桁
.	小数点を表示する桁
＋	正でも負でも，符号を表示する
−	負のときだけ，符号を表示する
＊＊	頭部の余った桁を＊で埋める
￥￥	数値の直前に￥を表示する
＊＊￥	上記の2つの働きの合成
，	整数部分を3桁ごとにコンマで区切る
^^^^	指数型の表示にする
！	文字データの最初の1文字だけを表示する
&　　&	文字データの表示桁数を指定する

のとなります．また，書式記号で指定した画面の幅の中に数値データを表示しきれない場合には，直前に％が表示されます．

以下に，各記号について例を示して説明を加えます．

〔例１〕

```
10  A＝123：B＝－123：C＝456.91
20  PRINT  USING "＃＃＃＃＃␣＃＃＃＃␣＃＃＃"; A, B, C
30  PRINT  USING "＃＃＃＃␣␣"; A, B
40  PRINT  USING "＃＃"; A
```

これを実行すると次のように出力されます．(上記の中の␣は空白です．)

20	書式仕様	＃＃＃＃＃	␣	＃＃＃＃	␣	＃＃＃	←これは説明です．画面には現われません．
	出力結果	１２３		－１２３		４５７	←これが画面に表示されます．
30	書式仕様	＃＃＃＃	␣␣	＃＃＃＃	␣␣		
	出力結果	１２３		－１２３			
40	書式仕様	＃＃					
	出力結果	％１２３					

＃で指定された幅（桁数）の中で，数値は右づめで表示されます．頭部の余った桁は空白となります．40番のように＃の幅が不足すると，％が現われます．

数値が負のときは－が直前に表示されます．正のときは＋は表示されません．

456.91は４捨５入されて457と表示されます．

30番では，ひとつの書式仕様がＡとＢに対して使われています．このように，書式仕様が不足した場合は，同じ書式仕様がくり返し使われます．

〔例２〕

```
10  A＝12.345: B＝6.7: C＝0.5678
20  PRINT USING "␣＃＃．＃＃＃"; A, B, C
```

書式仕様	␣＃＃．＃＃＃	␣＃＃．＃＃＃	␣＃＃．＃＃＃
出力結果	１２．３４５	６．７００	０．５６８

任意の位置に小数点１個を挿入することができます．小数点以下の余った桁には０が表示されます．

〔例３〕

```
10  Y＝1980：M＝5：D＝12：V＝123.45
20  PRINT USING "＃＃＃＃年␣＃＃月␣＃＃日␣ウマレ"; Y, M, D
30  PRINT USING "ジソク␣＃＃＃．＃␣km"; V
```

20	書式仕様	＃＃＃＃	年␣	＃＃	月␣	＃＃	日␣ウマレ
	出力結果	１９８０	年	５	月	１２	日　ウマレ
30	書式仕様	ジソク␣	＃＃＃．＃	␣km			
	出力結果	ジソク	１２３．５	km			

この例のように，引用符（"）で囲まれた書式仕様の中に書式記号以外の文字（空白も含む）が

あると，その文字がそのまま表示されます．

〔例4〕
```
10  A＝12：B＝－34：C＝56：D＝－78
20  PRINT USING "＋##␣＋##␣##＋␣##＋"; A, B, C, D
30  PRINT USING "##－␣##－"; C, D
```

20	書式仕様	＋##	␣	＋##	␣	##＋	␣	##＋
	出力結果	＋12		－34		56＋		78－
30	書式仕様	##－	␣	##－				
	出力結果	56		78－				

この例のように，書式記号＋を用いると，その桁に符号（＋または－）が表示されます．書式記号－を用いると，値が正の場合は符号桁は空白となります．

書式記号＋は#の直前または直後に付けることができます．書式記号－は#の直後に付けます．

〔例5〕
```
10  A＝1.2：B＝－34.5：C＝1234.5
20  PRINT USING "**##.#␣"; A, B, C
```

書式仕様	**##.#	␣	**##.#	␣	**##.#	␣
出力結果	***1.2		*－34.5		1234.5	

2個のアスタリスク（**）を先頭に付けると，余った桁が*で埋められます．桁が余らなければ，*は表示されません．

〔例6〕
```
10  A＝123
20  PRINT USING "¥¥######"; A
30  PRINT USING "**¥#####"; A
```

20	書式仕様	¥¥######
	出力結果	¥123
30	書式仕様	**¥#####
	出力結果	****¥123

2個の円記号（¥¥）を先頭に付けると，数字の直前に¥を1個表示します．

2個のアスタリスクと1個の円記号（**¥）を先頭に付けると，数字の直前に¥を1個表示し，さらに左側の余った桁が*で埋められます．

〔例７〕

```
10  A#＝12345678：B＝1234.5
20  PRINT USING "###########,␣######,.##";A#,B
```

書式仕様	##########,	␣	######,.##
出力結果	12,345,678		1,234.50

小数点が無い場合は#の直後に，また小数点がある場合は小数点の直前に，コンマ（,）を書くと，この例のように３桁ごとにコンマで区切って表示します．これにさらに，**，¥¥，**¥などの書式記号を組合わせて使うこともできます．

〔例８〕

```
10  A＝23456：B＝－0.00023456
20  PRINT USING "##.##∧∧∧∧␣";A,B
```

書式仕様	##.##∧∧∧∧	␣	##.##∧∧∧∧	␣
出力結果	2.35E+04		－2.35E－04	

４個の矢印（∧∧∧∧）を#の後に付けると，指数形式で表示します．有効数字は左づめで表示され，指数部はそれに合わせて調整されます．最左端の桁は符号を表示するために使われます．

なお，指数形式で表示する場合は，円記号（¥）やコンマ（,）を表示する書式指定は使えません．

〔例９〕

```
10  A$＝"XYZ"
20  PRINT USING "!";A$
30  PRINT USING "&␣␣␣&";A$
40  PRINT USING "&&";A$
```

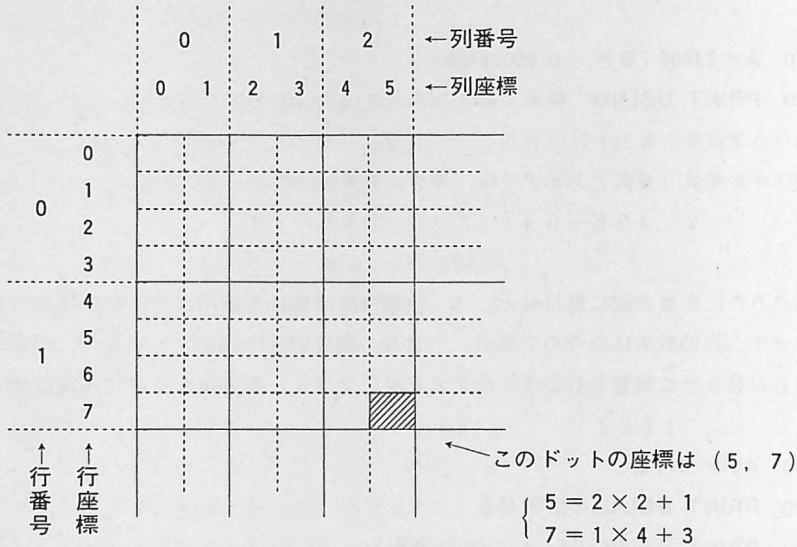
	文字データ	XYZ
20	書式仕様	!
	出力結果	X
30	書式仕様	&␣␣␣&
	出力結果	XYZ
40	書式仕様	&&
	出力結果	XY

書式記号として！を用いると，文字データの最初の１文字だけが出力されます．

n個の空白を挟んだ&（&␣␣…␣␣&）は，(n＋2）桁の画面領域に文字データを表示します．文字データは左づめされ，右側の余った桁は空白となります．また，(n＋2）桁の画面領域に入りきれない文字は表示されません．

8.4　ドットの出力

本節では，グラフィック表示すなわちドット（点）を画面に表示するための命令について説明します．

1個のドットの大きさは，キャラクタ表示における1文字の大きさの8分の1です．次図のように，1文字分の画面領域の横を2等分し，縦を4等分したものが1個のドットです．そして，各ドットの位置を指定するために番号を用いますが，この番号をドットの列座標，行座標と呼ぶことにします．画面の左上隅の位置の座標が0番です．

列や行の番号と座標との関係は次式で表わされます．

列座標＝列番号×2＋（0または1）
行座標＝行番号×4＋（0，1，2，または3）

画面上の指定した位置に1個のドットを表示するには，次のPSET文を用います．

PSET（〈列座標〉，〈行座標〉，〈機能コード〉）

機能コードは，COLOR文のとき同じ書き方をします．したがって，白黒モードかカラーモードかによってその働きが異なります．また，機能コードの指定を省略することができますが，省略した場合には，前のCOLOR文で指定した機能コードに従って動作します．なお，座標の指定が整数でないときは小数部分は切捨てられます．

〔例1〕　PSET（5，7，2）

このPSET文によって，上掲の図の位置（5，7）にドットが表示されます．機能コードは2が

指定されているので，白黒モードではブリンク，カラーモードでは赤色のドットとなります．

〔例2〕
```
10 FOR J=0 TO 19
20 PSET (J, J)
30 NEXT J
```

これを実行すると，画面の左上隅から斜めに，20個のドットが直線上に並んで表示されます．なお，画面に直線を書くには，もっと便利な命令があります．次項を参照してください．

PSET文でドットを書くと，そのドットが所属している1文字分の画面領域に表示されていた文字は消えてしまいます．また，1文字分の画面領域の中に2個以上のドットを書く場合は，機能コードは，後から指定したものが優先されます．いいかえれば，1文字分の画面領域の中の各ドットに，それぞれ異った機能を割当てることはできない，ということです．

画面上のある位置に表示されているドットを消すには，次のPRESET文を用います．

PRESET (〈列座標〉, 〈行座標〉)

〔例3〕 PRESET (5, 7)

これによって，前掲の図に表示されているドットが消えます．

画面上のある位置に，ドットが表示されているかいないかを調べるには，次のPOINT関数を使います．

POINT (〈列座標〉, 〈行座標〉)

指定した位置にドットが表示されていれば－1，表示されていなければ0が関数の値となります．

〔例4〕 100 IF POINT (5, 7) = -1 GOTO 200

8.5 直線や箱を書く

画面に直線あるいは箱（長方形）を書くために便利なLINE文を紹介します．このLINE文は8・1・4で説明したLINE文とは違います．直線や箱を書くのに，文字を用いる場合とドットを用いる場合とがあります．また指定の仕方によって，箱の中を文字あるいはドットで埋めることもできます．

(1) 文字を用いる場合

LINE (X_1, Y_1) - (X_2, Y_2), 〈文字の指定〉, 〈機能コード〉, 〈箱の指定〉

X_1, X_2は画面の位置の列番号であり，Y_1, Y_2は行番号です．文字の指定は，任意の文字を引用

符（″）で囲んで書きます．あるいはCHR＄関数を使って書いてもよろしい．機能コードはCOLOR文で使うものと同じです．したがって，白黒モードのときとカラーモードのときとではその働きが異なります．機能コードの指定は省略することができます．箱の指定は，BまたはBFを書きます．

直線を書くときは，箱の指定は書きません．この場合は，（X_1，Y_1）の位置と（X_2，Y_2）の位置とを結ぶ直線（線分）が，指定した文字を使って表示されます．

箱の指定をBと書くと，上記の直線を対角線とする箱（長方形の枠）が表示されます．箱の指定をBFと書くと，長方形の内部が指定した文字で埋められます．

〔例1〕
```
10 LINE (1, 1)-(5, 5), "*", 2
20 LINE (0, 0)-(40, 20), CHR$ (42), , B
30 LINE (6, 10)-(15, 18), "A", 4, BF
```

この例の10番のLINE文では，＊を使って直線を書きます．20番では箱（の枠）を書きます．文字の指定にCHR＄関数を使っており，機能コードの指定は省略されています．30番では箱の内部までAという文字で埋まります．

(2)　ドットを用いる場合

```
LINE (x1, y1)-(x2, y2), 〈ドットの指定〉, 〈機能コード〉, 〈箱の指定〉
```

x_1，x_2は画面の位置の列座標であり，y_1，y_2は行座標です．ドットの指定としては，PSETまたはPRESETのいずれかを書きます．PSETにすれば直線あるいは箱をドットで表示します．PRESETにすれば，表示されている直線あるいは箱を消します．その他のことは，上述の文字を用いる場合と同様です．

〔例2〕
```
10 WIDTH 80, 25
20 CONSOLE 0, 25, 0, 0
30 LINE (0, 0)-(159, 99), PSET, 0, B
```

これは画面のもっとも外側に，ドットで枠を書きます．

8.6　画面データの記憶と再出力

画面上に表示されているデータ（文字やドット）を配列内に記憶すること，あるいは，その記憶されたデータを画面の任意の場所に出力することができます．本節ではこの方法を説明します．

画面データを配列に記憶するために，GET@文を使います．それを再出力するために，PUT@文を使います．これらは指定の仕方によって，次の3つの形式があります．

A指定	G指定	取扱う対象	その他の機能
——	——	キャラクタ表示（文字）	出力時に色やブリンクなどを指定できる
——	G	グラフィック表示（ドット）	出力時に画面との演算を指定する
A	——	キャラクタ＋グラフィック表示	色やブリンクなども記憶する

8.6.1　キャラクタ表示の記憶と再出力

指定した画面領域に表示されているキャラクタ（文字）を配列に記憶させるために，次の形のGET@文を使います．

GET@（X_1，Y_1）－（X_2，Y_2），〈配列名〉

X_1，X_2は画面の位置の列番号であり，Y_1，Y_2は行番号です．すなわち，前節8·5(1)のLINE文で用いるものと同じ意味のものです．（X_1，Y_1）の位置と（X_2，Y_2）の位置とを結ぶ直線(線分)を対角線とする長方形の画面領域にあるキャラクタ表示を，配列内に記憶します．

配列名は，あらかじめDIM文で宣言しておきます．この配列は1次元のものとし，その大きさ（配列要素の個数）は次のようにします．

〈配列要素の個数〉＝〈記憶すべき画面領域の面積〉÷K──→割り切れないときは切り上げる．

ここで，画面領域の面積はキャラクタ（文字）単位で計ったものです．またKは，配列要素1個あたりのバイト数であり，具体的には次の値です．

配列が整数型のときK＝2，単精度型のときK＝4，倍精度型のときK＝8

1バイトに1文字ずつ記憶します．

例えば，3行×4列＝12文字を記憶させる場合，整数型配列A％を用いるとすれば，面積は12ですから，

〈配列要素の個数〉＝12÷2＝6　ですから次のように宣言すればよいことになります．

DIM　A%(5)……配列要素番号（添字）は0番から始まるので，6－1＝5となります．

上記のGET@文で記憶したものを画面に出力させるために，次の形のPUT@文を使います．

PUT@（X_3，Y_3）－（X_4，Y_4），〈配列名〉

X_3，X_4は画面の列番号であり，Y_3，Y_4は行番号です．（X_3，Y_3）の位置と（X_4，Y_4）の位置とを結ぶ直線を対角線とする長方形の画面領域に，配列内に記憶されているデータ（文字）を出力します．

〔例 1〕

```
10 DIM  A(2)  ←―配列宣言
20 PRINT  CHR$(12)  ←―画面消去
30 LOCATE  3, 2:PRINT "123456"   ←―┐画面に文字を書く
40 LOCATE  3, 3:PRINT "ABCDEF"   ←―┘
50 GET@ (3, 2)-(8, 3), A  ←―画面データをAに記憶する
60 PUT@ (12, 6)-(15, 8), A  ←―再出力する
```

この例では，記憶すべき画面領域の面積は，2行×6列＝12文字です．単精度型の配列Aを用いているので，

〈配列要素の個数〉＝12÷4＝3 ――→3－1＝2

ですから，DIM A(2)と宣言しています．

GET@文によって，配列内には次のように記憶されます．

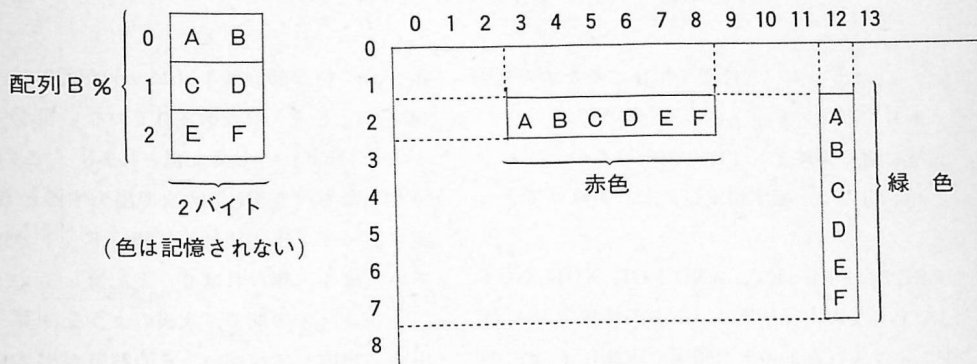
実行の結果，画面は次図のようになります．

〔例 2〕

```
10 DIM  B%(2)
20 CONSOLE , , 0, 1 ←―カラーモードを指定
30 PRINT  CHR$(12)  ←―画面消去
40 COLOR  2:LOCATE  3, 2:PRINT "ABCDEF" ←―赤色で文字を書く
50 GET@ (3, 2)-(8, 2), B%
60 COLOR  4:PUT@ (12, 2)-(12, 7), B%      ←―緑色で再出力する
70 COLOR  7:LOCATE  0, 0
```

この例では，記憶すべき面積は6文字です．整数型配列B％を用いているので，

〈配列要素〉＝6÷2＝3──→3－1＝2──→DIM B％(2)となります．

配列内の記憶状態および実行結果の画面は次図のようになります．

PUT@文の末尾に機能コードを付けることができます．次の形です．

PUT@ $(X_3, Y_3)-(X_4, Y_4)$，〈配列名〉，〈機能コード〉

機能コードの書き方は，COLOR文(8･1･3)で説明したのと同じであり，0～7の範囲で指定します．白黒モードかカラーモードかによって，その働きが異なることに注意してください．

〔例3〕 前掲の〔例2〕の60番のステートメントを次のように書いても同じ結果になります．

```
60  PUT@ (12, 2)-(12, 7), B%, 4
                              ↑
                              └──機能コードの指定
```

8.6.2 グラフィック表示の記憶と再出力

指定した画面領域に表示されているグラフィック表示（ドット）を配列に記憶させるために，次の形のGET@文を使います．

GET@ $(x_1, y_1)-(x_2, y_2)$，〈配列名〉，G

これは前項で紹介したGET@文と似ていますが，末尾にGが付いている点が違います．また，画面上の位置の数え方も違います．それにともなって，配列の大きさの計算法も異なります．

x_1，x_2はドット単位で数えた画面位置の列座標です．同様にy_1，y_2は行座標です．

必要な配列の大きさは次のように計算してきめ，あらかじめDIM文で宣言しておきます．

〈配列要素の個数〉＝(〈記憶すべき画面領域の面積〉÷8＋2)÷K──→小数点以下は切り上げる．

ここで，画面領域の面積はドット単位で計ったものです．Kは前項で述べたものと同じです．

ドット1個分を記憶するのに1ビットを使います．8ビットで1文字分（1バイト）になります．

上記のGET@文で記憶したものを画面に出力させるために，次の形のPUT@を使います．

PUT@（x_3，y_3）－（x_4，y_4），〈配列名〉，〈条件〉

x_3，x_4は出力すべき位置を指定するための列座標であり，同様にy_3，y_4は行座標です．

末尾に書く条件としては，次のうちのいずれかひとつを選んで，必ず指定します．省略はできません．

PSET，PRESET，AND，OR，XOR，NOT

PUT@文の実行結果は，この条件指定およびPUT@文実行直前の出力領域の状態によってきまります．GET@文を実行すると，ドットが表示されている部分は1（ビットがON）として記憶され，ドットが表示されていない部分は0（ビットがOFF）として記憶されます．このように記憶されたものをPUT@文で出力するとき，条件指定に従って，出力領域の状態（ドットが表示されていれば1，無ければ0）と記憶している状態（1か0）との間で，次表のような演算を1ビッドずつ対応して行ない，その結果が出力されます．

配列内の記憶状態	出力領域（PUT@文実行直前の状態）	PSET	NOT	PRESET	AND	OR	XOR
1	1	1	0	0	1	1	0
	0	1	0	0	0	1	1
0	1	0	1	1	0	1	1
	0	0	1	0	0	0	0

演算結果（PUT@文実行後の画面の状態）

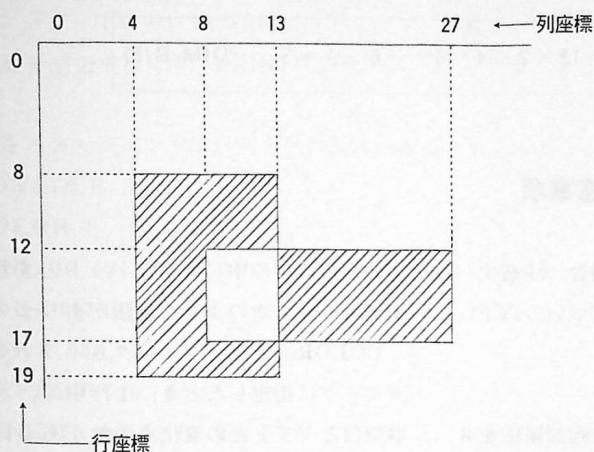
〔例〕

```
10  DIM  A%(8)
20  COLOR, , 1 : WIDTH  40, 25
30  PRINT  CHR$(12)
40  LINE (4, 8)-(13, 19), PSET, , BF  ←─箱を書く
50  GET@ (4, 8)-(13, 19), A%, G
60  PUT@ (8, 12)-(27, 17), A%, XOR
```

この例では，画面領域の面積は，（13－3）×（19－7）＝120ドットです．整数型配列A％を用いれば，

〈配列要素の個数〉＝（120÷8＋2）÷2＝8.5 ──→ 切り上げて9 ──→ 9－1＝8

したがって，DIM　A％(8)と宣言すればよいわけです．

これを実行すると，画面は図のようになります，PUT@文の条件としてXORを指定したので，2つの長方形が交わっている領域のドットが消えました．

8.6.3 文字・ドット混合表示の記憶と再出力

指定した画面領域に表示されている文字およびドット（混在）を，いっしょに配列に記憶したり，それを再出力したりするときは，次の形のGET@文あるいはPUT@文を使います．これを使うと，色やブリンク・リバースなどの状態も同時に配列に記憶したり再出力したりします．

```
GET@A (X1, Y1) - (X2, Y2), 〈配列名〉
```

```
PUT@A(X3, Y3) - (X4, Y4), 〈配列名〉
```

X，Yは画面の位置を指定する列番号，行番号です．すなわち8·6·1（キャラクタ表示の場合）と同じです．

配列の大きさは次のように計算して求められます．つまり，キャラクタ表示の場合の2倍の大きさです．

〈配列要素の個数〉＝〈記憶すべき画面領域の面積〉×2÷K

ここで，領域の面積はキャラクタ（文字）単位で計ったものです．Kは前出のものと同じです．

〔例〕

```
10  DIM  B(5)
      ∫
100  GET@A(3, 2) - (8,3), B
110  PUT@A(12,6) - (15,8), B
```

この例では，画面領域の面積は12文字です．単精度型の配列Bの大きさは次のように計算されます．

〈配列要素の個数〉＝12×2÷4＝6──→6 －1＝5──→DIM B(5)

8.7　画面表示における注意事項

画面に文字やドットを表示する場合，注意すべき点や制限事項があります．これについて以下に述べます．

(1)　1文字分の画面領域の性質

すでに述べたように，1文字分の画面領域を8等分したものがドット1個分になります．1文字分の領域の中に，文字とドットとを混在で同時に表示することはできません．

例えばある位置に1文字が表示されているとき，その1文字分の領域の中に座標を指定してドットを出力すると，その文字は消えて，ドットだけが表示されます．また逆に，ある位置にドットが表示されているとき，そのドットを含む1文字分の領域に文字を出力すると，ドットは消えて，文字だけが表示されます．

次に色についてですが，1文字分の領域の中に複数個のドットを表示するとき，それらの各ドットを違った色にすることはできません．後から出力したドットの色に揃えられます．

(2)　1行中の文字とドットの混在

画面の1行の中に文字とドットとを混在で表示するときは，次のような制限があります．

COLOR文でグラフィックスイッチをキャラクタモードに指定したとき，1行中で，文字もしくは空白とドットとの変化点の数がある限度を超えると，それより右側の部分にはドットは表示できなくなります．

またグラフィックモードが指定されているとき，1行中で，ドットもしくは空白と文字との変化点の数がある限度を超えると，それより右側の部分には文字は表示できなくなります．

ここで変化点の数の限度というのは，白黒モードの場合は20カ所，カラーモードの場合は19カ所です．

(3)　1行中の色の変化

画面の1行中で，色（空白も含む）の変化点の数が19カ所を超えると，それより右側の部分では，色の変化ができなくなります．

以上のような制限条件があるので，非常に複雑な表示をしようとする場合は注意が必要です．

理解度テスト

(1)　次の文章の空欄［　　］を埋めなさい．

① 電源投入時においては，画面の左端の列番号は［　　］番であり，右端の列番号は［　　］番である．

② WIDTH文で指定できる桁数は,最小［　　］桁，最大［　　］桁である．また行数は，最小［　　］行，最大［　　］行である．

③ 画面のスクロール範囲の指定は［　　］文によって行なう．

④ 画面のヌルキャラクタコードは［　　］文によって指定する．

⑤ 白黒モードかカラーモードかの指定は［　　］文によって行なう．

⑥ 画面下段のファンクションキーの表示の有無は［　　］文によって指定する．

⑦ COLOR文における機能コードの指定には［　　］の範囲の数字を用いる．

⑧ PSET文は，画面に［　　］を表示するために用いる．

⑨　カラーモードにおいて，機能コード2は［　　］色，6は［　　］色を指定する．

⑩　画面を消去するには，PRINT CHR$（［　　］）を用いる．

(2)　次の各ステートメントは，何をどのように指定しているのか，説明しなさい．

①　CONSOLE 5，19，0，0

②　COLOR 6，0，0

③　LINE 10，2

④　LOCATE 10，2

⑤　PSET（10，2）

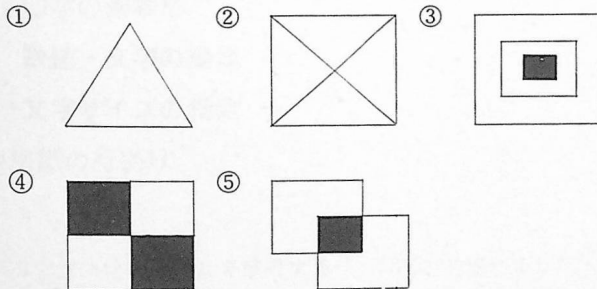
⑥　PRESET（10，2）

⑦　LINE（0,0）－（10,10），"A"，，BF

⑧　LINE（0,0）－（10,10），PSET，，BF

⑨　GET@（0，0）－（10，10），B

⑩　PUT@（0，0）－（10，10），P，PSET

(3)　ドットを用いて，画面に下記のような図形を表示するプログラムを作りなさい．図形の大きさや表示位置は自分で自由にきめなさい．

第9章　プリンタによる印字

PC－8001にプリンタ（印字装置）を接続することによって，用紙に印字することができます．印字する内容はデータおよびプログラムです．

プログラムを用紙に印字するにはLLISTという命令を使います．これについてはすでに第2章(2・3・2)で述べました．本章では，データ（数値および文字）を印字する方法について解説します．

なお，本章における説明は下記の装置を用いることを前提としています．

PC－8021：80桁ドットインパクトプリンタ

9・1　プリンタのあらまし

プリンタ（PC－8021）による印字について，基礎的な事項を本節で述べておきます．

(1)　印字できる文字の種類

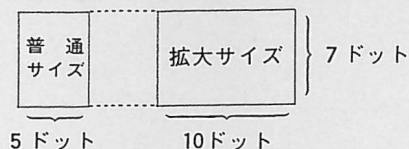
用紙に印字できる文字は，数字・英字（大文字・小文字）・カサ文字・特殊文字です．グラフィック文字（図形文字および漢字）は印字できません．グラフィック文字とは，キーボードでいえば，GRPHキーを使って指定する文字です．第6章（6・1)に掲げた文字コード表でいえば，上位4ビットが8, 9, E, Fのものがグラフィック文字です．また上位4ビットが2, 3, 4, 5, 6, 7, A, B, C, Dのものが印字可能な文字です．

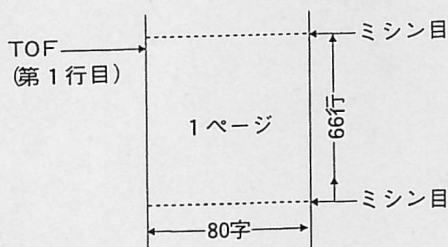
(2) 文字の印字サイズ

用紙に印字される文字のサイズ（大きさ）は2種類あり，プログラムの中で指定することができます．

その1は普通サイズの文字です．普通サイズ文字は用紙の1行に80文字まで印字できます．通常のデータの印字にはこれを用います．

その2は拡大サイズの文字です．これは1文字の横幅が普通サイズ文字の2倍です．そこで横倍文字とも呼ばれます．縦の長さは普通サイズと同じです．拡大サイズ文字は用紙の1行に40文字まで印字できます．

拡大サイズを指定する方法は次節で述べます．

(3) 印字用紙

通常用いる標準型の印字用紙は，横幅は80字／行のサイズになっており，縦方向は66行ごとにミシン目で区切られています．この（80字×66行）を1ページと考えます．1ページの最上行をTOF (Top Of Form) と呼びます．

(4) 内部機能制御スイッチ

プリンタの内部に4連の機能制御スイッチがあります．これをONにするかOFFにするかによって，プリンタの動作に若干の相違がありますが，その詳細な説明は省略します．次節以降では，このスイッチは下記のようにセットされているものとします．

スイッチ番号→	1	2	3	4
セットの状態→	OFF	OFF	ON	ON

9・2 数値・文字の印字

数値データや文字データをプリンタで印字するためのステートメント，および文字のサイズを指定する方法について，本節で解説します．

9・2・1 印字の基本形

プリンタで印字をするには，次のLPRINT文が基本となります．

```
LPRINT 〈出力並び〉
```

これはディスプレイ装置の画面に出力する場合のPRINT文と非常によく似ています．出力並びの書き方やその意味は，PRINT文の場合と同様ですから，第8章（8·2·1）を参照してください．ただし，そこの〔例4〕に掲げたような機能文字（画面消去・カーソル移動・その他）については，画面出力の場合とプリンタの場合とでは異なります．

また8·2·2（3）に掲げたTABおよびSPCは，LPRINT文の中でも使うことができます．

〔例〕

```
10 FOR N=1 TO 10
20 LPRINT N, N*N, SQR(N)
30 LPRINT ←── 1行空白とする
40 NEXT N
```

これを実行すると，用紙上に次のように印字されます．

```
1             1             1
                                  ←1行空白
2             4             1.41421
3             9             1.73205
4             16            2
5             25            2.23607
6             36            2.44949
7             49            2.64575
8             64            2.82843
9             81            3
10            100           3.16228
|←── 14桁 ──→|←── 14桁 ──→|
```

9·2·2　数値・文字の編集

画面出力の場合と同様に，プリンタにおいても数値・文字データを編集して印字することができます．このために次のLPRINT USING文を用います．

```
LPRINT USING "〈書式仕様〉"; 〈出力並び〉
```

書式仕様や出力並びの書き方は画面出力の場合と同様ですから，第8章（8·3）を参照してくだ

さい.

〔例〕
```
10 FOR N＝1 TO 10
20 LPRINT USING"###␣#####␣###・####";N,N*N,SQR(N)
30 NEXT N
```

これを実行すると，用紙上に次のように印字されます．前項で示した例と比較してみてください．

```
 1      1    1.0000
 2      4    1.4142
 3      9    1.7321
 4     16    2.0000
 5     25    2.2361
 6     36    2.4495
 7     49    2.6458
 8     64    2.8284
 9     81    3.0000
10    100    3.1623
```

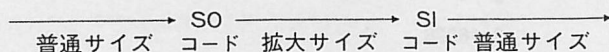
9･2･3 文字サイズの指定

プリンタで印字する文字の大きさ（普通サイズか拡大サイズか）を指定するには，下記の機能文字を用います．

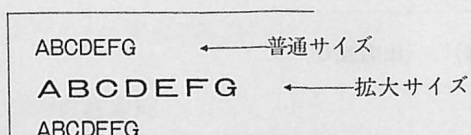
拡大サイズ指定←CHR$(14)…これはSOコードと呼ばれる．(Shift Out)

普通サイズ指定←CHR$(15)…これはSIコードと呼ばれる．(Shift In)

LPRINT文の中にSOコードがあると，それ以降は拡大サイズで印字されます．そしてこの状態は，SIコードが現われるまで続きます．SIコードが現われると，それ以降は普通サイズで印字されます．すなわち，SOコードやSIコードは、文字サイズを切り換える働きをするものです．

```
──────→ SO ──────→ SI ──────→
普通サイズ  コード 拡大サイズ コード 普通サイズ
```

〔例〕
```
10 X$="ABCDEFG"
20 LPRINT X$:LPRINT
30 LPRINT CHR$(14)  ───→拡大サイズを指定
40 LPRINT X$:LPRINT
50 LPRINT CHR$(15)  ───→普通サイズを指定
60 LPRINT X$
```

これを実行すると次図のように印字されます．

```
ABCDEFG      ←──普通サイズ
A B C D E F G  ←──拡大サイズ
ABCDEFG
```

9・3　印字用紙の行送り

印字用紙を縦方向に移動させることを，行送りあるいは紙送りといいます．行送りは1方向だけであり，逆方向の移動はできません．

行送りをさせるために次の機能文字を使うことができます．

TOFまで移動←── CHR$(12)…これはFFコードと呼ばれる．

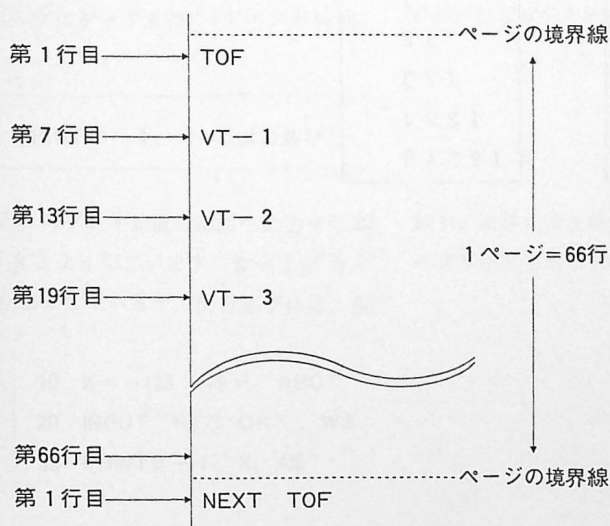
タブ位置まで移動←── CHR$(11)…これはVTコードと呼ばれる．

LPRINT文の中にFFコードがあると，これにより，用紙は次のページのTOF(第1行目)まで自動的に送られます．これを，ページ替えとか改ページと呼びます．ページ替えをしてから次のLPRINT文を実行すれば，新しいページの第1行目に印字することができます．

ディスプレイ装置の画面への出力の場合は，PRINT文の中のCHR$(12)は，画面を消去してカーソルをホームポジションへ移動する働きをしますが，これは画面のページ替えと考えることもできるでしょう．

LPRINT文の中にVTコードがあると，これにより，用紙は次のタブ位置まで自動的に送られます．タブ位置は，TOF(第1行目)から6行ごとにあらかじめ設定されています．

〔例1〕
```
10 LPRINT CHR$(12)"TOF"
20 FOR J=1 TO 3
30 LPRINT CHR$(11)"VT—";J
40 NEXT J
50 LPRINT CHR$(12)"NEXT TOF"
```

これを実行すると図のように印字されます．

これは，前節（9·2·1）の例のプログラムに，5　6，7番のステートメントを追加したものです．まずページ替えをして，スウヒョウという文字を拡大サイズで印字します．

〔例2〕
```
 5  LRRINT  CHR$(12)：LPRINT
 6  LPRINT  CHR$(14)"␣スウヒョウ"CHR$(15)
 7  LPRINT
10  FOR  N＝1  TO  10
20  LPRINT  N, N*N, SQR(N)
30  LPRINT
40  NEXT  N
```

理解度テスト

(1)　次の文章の空欄［　］を埋めなさい．

①　用紙に印字できる文字には，［　］サイズと［　］サイズの２種類がある．

②　CHR＄(14) は［　］サイズを指定し，CHR＄(15) は［　］サイズを指定する．

③　CHR＄（［　］）はTOFまでの紙送りを指定し，CHR＄（［　］）はタブ位置までの紙送りを指定する．

④　記憶装置内にあるプログラムを用紙に即字するには［　］という命令を用いる．

⑤　標準型の印字用紙の１ページは［　］行分である．

⑥　１ページの最上行を［　］と呼ぶ．

(2)　下記のように用紙に印字するプログラムを作りなさい．

①
```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

②
```
        1
      1 2
    1 2 3
  1 2 3 4
1 2 3 4 5
```

第10章　カセットテープの使い方

プログラムをカセットテープに記録する方法については，すでに第２章（2·3·5）で述べました．本章では，データをテープに記録したりそれを読み込んだりする方法について説明します．

ステートメントとしてはPRINT＃文およびINPUT＃文を用いますが，この両者の対応関係を適切にしなければなりません．また，文字データをテープに書き込む場合には，注意を要する点があります．以下の説明を読んでください．

10·1　テープへのデータの書き込み

カセットテープにデータを書く（データをsaveする）には次のPRINT＃文を用います．

```
PRINT＃－１，　〈出力並び〉
```

これは，ディスプレイ装置の画面へ出力する場合のPRINT文とよく似ています．＃－１がカセットテープを意味しています．出力並びには，変数名・定数・式を書くことができます．数値型でも文字型でもよろしい．

〔例１〕

```
10  X＝－123：A$＝"ABC"
20  INPUT "REC  OK"；W$
30  PRINT＃－１，X，A$
```

これを実行すると，20番のINPUT文によって「REC OK？」が画面に表示されます．そこで，カセットテープレコーダのキーを押して録音状態にします．それから，キーボードから任意の文定をW$へKey-inします．すると，30番のPRINT＃文によって，Xの値（－123）とA$の値（ABC）とがテープに書き込まれます．

もしテープの先頭からデータを書こうとするならば，レコーダを録音状態にする前に，テープの巻き戻しをしておくことが必要です．

20番のINPUT文は，テープレコーダを操作するための時間かせぎです．W$には特別な意味はありません．

〔例2〕

```
10  INPUT N：DIM A(N)
20  FOR J＝1  TO N          } N個のデータを配列Aへ入力
30  INPUT A(J) ：NEXT J
100 A(0)＝N
110 INPUT "REC OK"； W$
120 FOR J＝0  TO N
130 PRINT＃－1，A(J)        } データ件数(N)とN個のデータをテープに書き込む
140 NEXT J
```

これは，配列内のN個のデータをテープに書き込みます．その先頭にデータ件数(N)も書き込んでいます．

ところで，文字型データをテープに書き込む場合ちは，下記の点に注意が必要です．例を見てください．

〔例3〕

```
100 A$＝"ABC"：B$＝"DEF"
110 PRINT＃－1，A$，B$
120 PRINT＃－1，A$；B$
```

110番のPRINT＃文では，A$とB$とはコンマ(，)で区切られています．この場合はとくに問題はありません．A$の値（ABC）とB$の値（DEF）とは分離されてテープに書かれます．そしてこのテープを読むときも，ABCとDEFとは別のデータとして読まれます．つまり2個のデータがあるものとして読まれます．

120番のPRINT＃文では，A$とB$とはセミコロン(；)で区切られています．この場合は問題が生じます．A$の値とB$の値とは分離されず，ABCDEFというように連続した文字列としてテープに書かれます．このテープを読むときは，ABCDEFという1個のデータとして読まれます．つまり，テープ上には1個のデータしかないものとして読まれます，

〔例4〕

```
200 A$＝"␣␣ABC"：B$＝"DE，FG"
210 PRINT＃－1，A$
220 PRINT＃－1，CHR$(34)；A$；CHR$(34)
230 PRINT＃－1，B$
240 PRINT＃－1，CHR$(34)；B$；CHR$(34)
```

210番のPRINT＃文でテープに書かれたデータを読むと，A$の文字列の先頭の空白は読まれま

せん．つまり，200番でA＄＝〝ABC〞としたのと同じ結果になります．

220番ではA＄の両側にCHR＄(34)があります．これは引用符(〞)を意味します．34は引用符の文字コードです．220番のPRINT＃文でテープに書かれたデータを読むと，A＄の先頭の空白も文字列の構成要素として読まれます．

230番のPRINT＃文でテープに書かれたデータを読むと，DEとFGとは分離されて別個のデータとして読まれます．つまり，テープ上に2個のデータがあるものとして読まれます．

240番ではB＄の両側にCHR＄(34)があります．このようにすれば，B＄の内容は分離されず，EとFの間にあるコンマ(,)も文字列の構成要素として読まれます．

以上のように，PRINT＃文でテープに書こうとする文字列の先頭に意味のある空白が含まれている場合や，文字列の中にコンマが含まれている場合は，CHR＄(34)を用いることが必要です．

10·2　テープからのデータの読み込み

PRINT＃文でカセットテープに書いたデータをコンピュータの記憶装置の中へ読み込む（データをloadする）には，次のINPUT＃文を用います．

```
INPUT#－1, 〈入力並び〉
```

入力並びとは，テープから読み込んだデータを記憶すべき場所を指定するものであり，変数名で表わします．変数名が複数個のときはコンマ(,)で区切ります．

INPUT＃文の入力並びは，PRINT＃文の出力並びと対応していなければなりません．いいかえれば，それぞれの並びの個数，順序，およびデータの型（数値型か文字型か）が一致していなければなりません．これらが一致していれば，出力並びの変数名と入力並びの変数名とは異なっていてもかまいません．

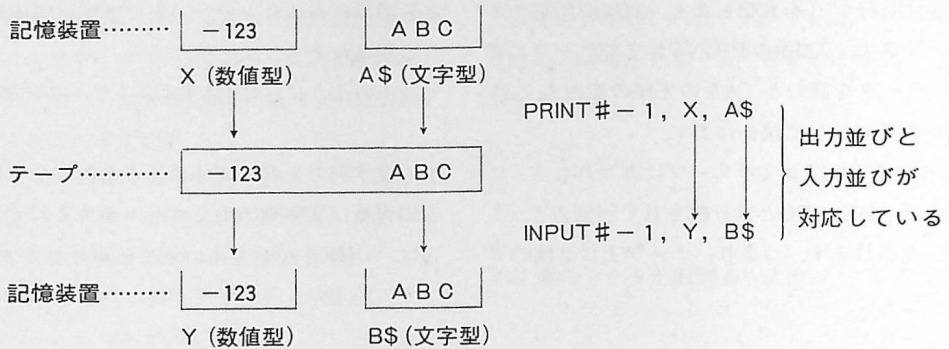
〔例 1〕

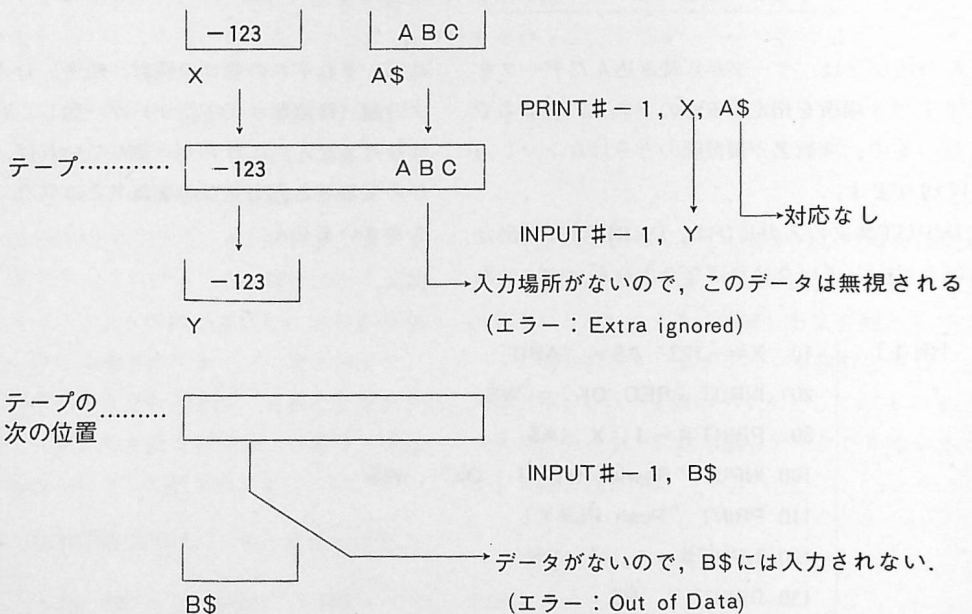
```
10  X=－123：A$= 〝ABC〞
20  INPUT 〝REC OK〞; W$
30  PRINT#－1, X, A$
100 INPUT 〝Rewind tape !  OK〞; W$
110 PRINT 〝Push PLAY !〞
120 INPUT#－1, Y, B$
130 PRINT Y, B$
```

このプログラムの前半(30番まで)は，前節(10·1)の〔例 1〕と同じです．これによってテープには，－123およびABCという2個のデータが書かれます．ここでは，このデータはテープの先頭から書かれたものとします．

100番のINPUT文は，テープの巻き戻しをするための時間かせぎです．W＄には特別な意味はありません．110番のPRINT文によって画面に「Push PLAY !」が表示されたら，テープレコーダのキーを押して，再生状態にします．

120番のINPUT＃文によってテープが読まれ，入力並びの変数Ｙには－123が入り,変数Ｂ＄はABCが入ります．130番のPRINT文によって，それらが画面に表示されます．

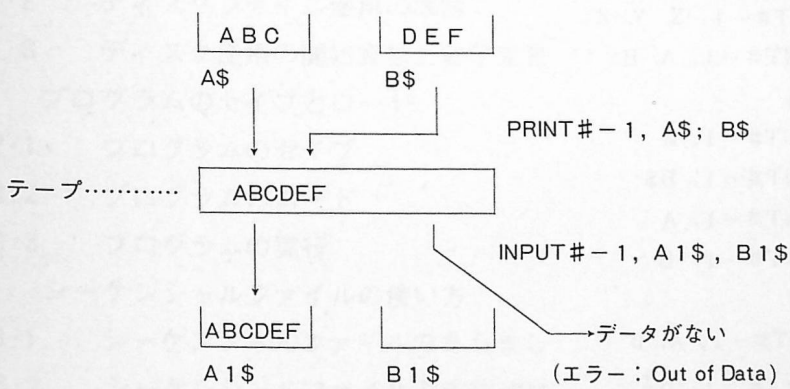
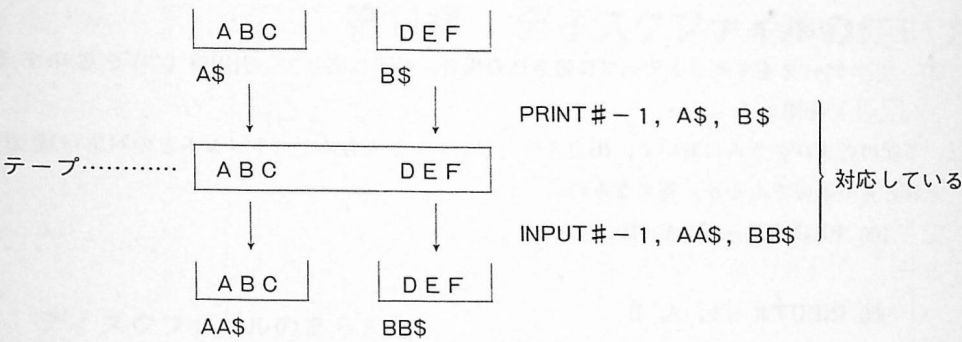
以上の状態を図示すると次のようになります．

上記のプログラムの120番のステートメントを２つに分けて下記のようにすると，出力並びと入力並びとが対応しなくなり，データをうまく読み込むことができなくなります．次図を見てください．

```
120 INPUT＃－１，Ｙ
121 INPUT＃－１，Ｂ＄
```

〔例２〕

```
200 INPUT＃－１，Ｋ
210 FOR Ｊ＝１ TO Ｋ
220 INPUT＃－１，Ｂ（Ｊ）
230 NEXT Ｊ
```

これは，前節（10·1）の〔例2〕でテープに書いたデータを，配列Bの中へ読み込むものです．

〔例3〕
```
200 INPUT＃－1，AA$，BB$
210 INPUT＃－1，A1$，B1$
```

この２つのINPUT＃文を使って，前節の〔例3〕でテープに書いたデータを読むと，次図のようになります．

〔例4〕　前節の〔例4〕でテープに書いたデータを読むものとすると，次のようになります．

```
310 INPUT＃－1，AA$ ──→ ABC     ………先頭の空白は無視される
320 INPUT＃－1，A1$ ──→ ␣␣ABC
330 INPUT＃－1，BB$ ──→ DE      ………FGは無視される
340 INPUT＃－1，B1$ ──→ DE，FG
```

理解度テスト

(1) 次の文章の空欄 □ を埋めなさい.

① プログラムをカセットテープに書き込むには □ という命令を用い，またそれを記憶装置へ読み込むには □ という命令を用いる.

② データをカセットテープに書き込むには □ 文を用い，またそれを記憶装置へ読み込むには □ 文を用いる.

③ 文字データをカセットテープに書き込む場合，必要に応じて，引用符（″）を意味するCHR＄（□）を用いる.

(2) 下記の各プログラムにおいて，出力ステートメントと入力ステートメントとの対応は適切であるか，それとも不適切であるか，答えなさい.

①
```
100 PRINT#－1，A，B
      ∫
500 INPUT#－1，A，B
```

②
```
100 PRINT#－1，A，B，C
      ∫
500 INPUT#－1，X，Y，Z
```

③
```
100 PRINT#－1，A，B$
      ∫
500 INPUT#－1，A
510 INPUT#－1，B$
```

④
```
100 PRINT#－1，A
110 PRINT#－1，B
      ∫
500 INPUT#－1，A，B
```

⑤
```
100 PRINT#－1，A$
      ∫
500 INPUT#－1，A
```

第11章　ディスクファイルの使い方

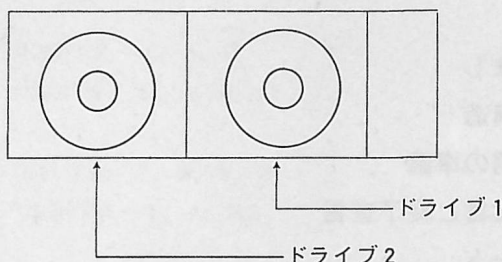
PC-8001には，下記のものを外部記憶装置として接続することができます．

デュアルミニディスク装置 PC-8031および PC－8032

この装置で用いる記録媒体をミニフロッピーディスクといいます．これはフレキシブルディスクとも呼ばれ，薄い円板です．この円板の表面に，磁気的な方法で，多量の情報を記憶させることができます．

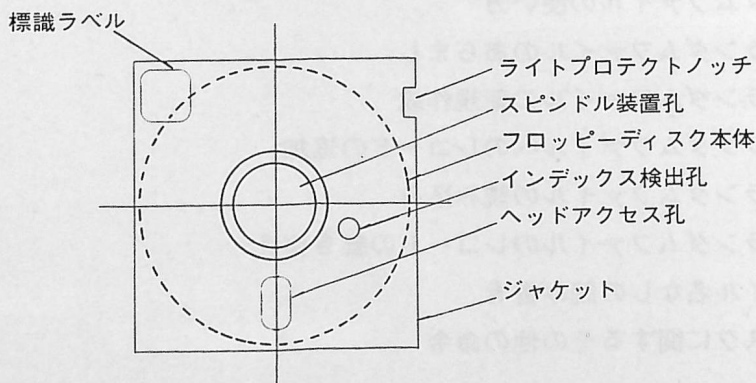
以下の説明では，上記の装置のことを単に「ディスク装置」と呼び，また記録媒体（円板）のことを単に「ディスク」と呼ぶことにします．

1台のディスク装置には，ディスクをセットする場所が2カ所あり，これを「ドライブ」といいます． PC－8031のドライブには1，2番，また PC－8032のドライブには3，4番という番号が付いています．これをドライブ番号といいます．

ドライブにディスクをセットして読み書きを行ないます．したがって，装置は1台だけでも，ディスクを多数枚用意すれば非常に多くの情報（データやプログラム）を記録し保存することができます．

11·1　ディスクファイルのあらまし

本節では，ディスク装置を利用するに当って必要な予備知識として，ディスクファイルの構造や準備作業あるいはディスクの使用宣言などについて説明します．

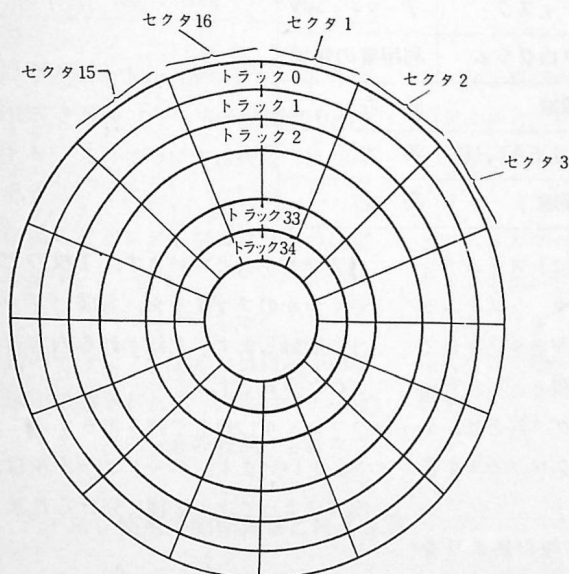
11·1·1　ディスクファイルの構造

ディスク（円板）は，それを保護するための4角形のジャケットに入れられており，次図のような形状になっています．

この円板に，片面倍密度という方式で情報を記録します．記録面は（眼には見えませんが）多数の記録場所に分割されています．

まず，同心円によって35個のリング状の部分に仕切られます．このそれぞれを「トラック」と呼びます．トラックには0番から34番までの番号(トラック番号）がついています．

また放射状に16個の部分に仕切られます．これを「セクタ」と呼び，1番から16番までの番号(セクタ番号）がついています．1セクタには256文字（バイト）の情報を記録することができます．

また，8セクタをひとまとめにしたものを「クラスタ」と呼びます．第0トラックの第1セクタから第8セクタまでが第0クラスタです．続いて，第0トラックの第9セクタから第16セクタまでが第1クラスタです．最後は，第34トラックの第9セクタから第16セクタまでの第69クラスタです．

次上のことをまとめると次図のようになります．

ディスク1枚＝35トラック＝70クラスタ＝560セクタ＝143360バイト

1トラック＝2クラスタ＝16セクタ

1クラスタ＝8セクタ

1セクタ＝256バイト

ディスクの読み書きは1セクタを最小単位として行なわれます．読み書きを行なうべき位置は，トラック番号とセクタ番号とによって指定することができます．この番号指定を用いて読み書きする方法については，後節（11·5）で述べますが，通常はこのような方法は用いません．すなわち，ディスク利用者は通常は，トラック番号やセクタ番号を意識する必要はないのです．

ではどうするのかというと，通常のプログラムでは，後述する「ファイル名」を用いてディスクファイルを取扱うのです．このファイル名に基いて，具体的にトラック番号やセクタ番号をきめて指定する仕事は，コンピュータ（BASICシステム）がやってくれるようになっているのです．つまり，利用者の負担をなるべく軽くするために，ディスクファイルを管理する仕事をコンピュータが担当してくれるようになっているのです．管理は1クラスタを最小単位として行なわれます．

このファイル管理の仕事をするために，コンピュータは，次の２つの配置表を用います．

① ディレクトリ（Directory）

② FAT

ディレクトリは，１枚のディスクに記録されているすべてのファイルの名前，性格および格納位置の表です．

FATは，１枚のディスクの中の記憶場所の状態（どのクラスタが使用中か未使用か）を示す表です．

ディスク１枚ごとに，そのディレクトリおよびFATが，特定の場所（次表参照）に書き込まれています．

トラック番号	トラックの割り当て	
	システムディスク	データディスク
0～2	システムプログラム	利用者の領域
3～17	利用者の領域	同　左
18	ディレクトリ,FAT,ID	同　左
19～34	利用者の領域	同　左

上表に示すように，ディスクにはシステムディスクとデータディスクとがあります．システムディスクとは，システムプログラムが書き込まれているディスクであり，ディスク装置とともに利用者に提供されます．システムプログラムとは，上述のようにコンピュータがディスクファイルを管理するためのプログラムです．

さて，一定の形に整理された情報の集まりを「ファイル」といいます．１枚のディスクには，いくつかのファイルを，名前（ファイル名）をつけて記録します．記録される内容はデータまたはプログラムです．

ファイルに対して読み書きすることを「アクセス」といいます．データファイルは，アクセスの仕方によって次の２種に分けられます．

① シーケンシャルアクセスファイル（略して，シーケンシャルファイル）

② ランダムアクセスファイル（略して，ランダムファイル）

この①については本章の11·3で，また②については11·4で解説します．またプログラムファイルについては11·2で解説します．

11·1·2　ディスクファイル使用の準備

ディスクファイルを使用するに当っては，まず最初に，次の２つの準備作業が必要です．

① システムプログラムのスタート

② ディスクのフォーマッティング

本項ではこれらについて説明します．

(1) システムプログラムのスタート

前項で述べたように，コンピュータがディスクファイルを管理するためのプログラム，すなわち

システムプログラムがシステムディスクに書かれています．このシステムプログラムを PC－8001の記憶装置に入れて，その活動をスタートさせることが必要です．これは次のようにして行ないます．

まずディスク装置（PC－8031）の電源を入れます．そして，システムディスクをディスク装置のドライブ1にセットします．次に PC－8001の電源を入れて，リセットボタンを押します．

すると，ディスクがアクセスされて，システムプログラムが PC－8001の記憶装置に読み込まれます．それから，ディスプレイ装置の画面に次のようなメッセージが表示されます．

```
DISK VERSION
How many files (0－15)?
```

そこで，15以下の数をKey-inします．この数は「同時にOPENされるファイルの個数の限界」を示すものですが，これについては後節（11・3）で説明を加えます．

以上の手順によってシステムプログラムの活動がスタートされます．

(2) ディスクのフォーマッティング

購入した新品のディスク，あるいは何らかの原因によって記録が破壊されたディスクに対しては，フォーマッティング（初期状態の設定）を行なうことが必要です．フォーマッティングにより，そのディスクにディレクトリやFATなどを記録することができるように準備されます．

フォーマッティングは，FORMATコマンドおよびシステムプログラムに含まれているformatプログラムを用いて行ないます．

なお，フォーマッティングを行なうと，そのディスクに記録されていたファイルはすべて消去されます．

11・1・3 ディスク使用の開始宣言と終了宣言

ディスクをディスク装置にセットして使い始めるときは，必ずその旨をコンピュータに知らせなければなりません．これを，ディスクの使用開始宣言といいます．このために次の命令を用います．

```
MOUNT <ドライブ番号>
```

〔例1〕 MOUNT 2

この例では，2番のドライブにセットしたディスクについて，使用開始の宣言をしています．この宣言をすると，ディスクに書かれている配置表が PC－8001の記憶装置に読み込まれます．そして，そのディスクが使用可能な状態になります．

使用が終わったディスクをドライブから取り出す前に，そのディスクの使用終了宣言をしなければなりません．このために次の命令を用います．

```
REMOVE <ドライブ番号
```

〔例2〕 REMOVE 2

この例では，2番のドライブにセットしてあるディスクについて，使用終了の宣言をしています．この宣言をすると，更新された最新の配置表がディスクに書き込まれます．使用終了宣言をしないでディスクをドライブから取り出すと，そのディスク内のファイルは使用不可能な状態になるので，厳重な注意が必要です．

11·2　プログラムのセイブとロード

作成したプログラムをディスクに記録すること（セイブ），またそのプログラムをディスクからコンピュータの記憶装置内へ読み込むこと（ロード）ができます．本節ではこれについて説明します．

すでに第2章で述べたように，プログラムのセイブやロードにはカセットテープを用いることができます．しかし，ディスクを用いれば，セイブやロードの時間を非常に短縮することができ，また信頼性も向上します．

11·2·1　プログラムのセイブ

PC－8001の記憶装置内にあるBASICプログラムをディスクにセイブするには，次のコマンドを用います．

```
SAVE　"〈ファイル名〉"
```

〔例〕 ①　SAVE "ABC"
　　　 ②　SAVE "1：ABC"
　　　 ③　SAVE "2：ABC"

ファイル名の長さは6文字以内とします．上記の例ではABCがファイル名です．

上例の②と③では，ファイル名の先頭に1：や2：が付いていますが，これはドライブ番号です．ドライブ番号が1のときは1：を省略することができます．上例の①がそうです．

したがって，引用符（"）の中の書き方は，詳しくいえば次の形になります．

"〈ドライブ番号〉：〈ファイル名〉"

本章での以下の説明では，これを単に次のように示します．ですから，2，3，4番のドライブを用いる場合には，先頭にドライブ番号を指定するのを忘れないように注意してください．

"〈ファイル名〉"

SAVEコマンドで指定したファイル名と同一のファイルがすでにディスクの中に存在している場合には，既存のファイル内容は消去されて，新しい内容がセイブされます．

さて，上記のSAVEコマンドでセイブした場合，プログラムはバイナリ形式と呼ばれる特別な表現形式でディスクに記録されます．これは，ディスクの記録場所を節約するために圧縮した表現形式です．通常はこれを用いればよいのです．

ところで，SAVEコマンドには次のような形もあります．これをA型と呼ぶことにします．

```
SAVE "〈ファイル名〉", A
```

A型でセイブすると，プログラムは，バイナリ形式ではなく，アスキー形式でディスクに記録されます．アスキー形式はバイナリ形式よりもディスクの記録場所を多く必要としますが，次に述べる「マージ」を行なう場合にはアスキー形式のセイブを用います．

記憶装置内にあるプログラムと，ディスクに記録されているプログラムとを合成して，1本のプログラムにまとめることができます．これをプログラムのマージといいます．このために次のコマンドを用います．

```
MERGE "〈ファイル名〉"
```

マージに用いるプログラムファイルは，アスキー形式でディスクに記録されていなければなりません．マージは行単位で行なわれます．マージによって合成されたプログラムは，記憶装置の中にできます．

11·2·2　プログラムのロード

ディスクに記録されているプログラムを PC－8001の記憶装置内にロードするには，次のコマンドを用います．

```
LOAD "〈ファイル名〉"
```

〔例〕 LOAD "ABC"

上記の LOADコマンドを実行すると，その直前まで記憶装置内にあったデータやプログラムは，すべて消去されます．また，直前までOPENされていたデータファイルは，すべてCLOSEされます．OPENやCLOSEについては次節で説明します．

なお，LOADコマンドには別の形（R型）がありますが，それは次項で紹介します．

11·2·3　プログラムの実行

ディスクからプログラムをロードして，そのプログラムを実行させるには，次の2つのコマンドを順に与えればよいわけです．

① LOAD "〈ファイル名〉"……ロードする　　② RUN ………実行を開始する

ところで，この①と②の働きを兼ね備えたコマンドがあります．次の形です．

RUN "〈ファイル名〉"

このRUNコマンドを与えると，ディスクからプログラムをロードして，すぐにそのプログラムの実行を開始します．

なお，RUNコマンドには次のような形もあります．これをR型と呼ぶことにします．

RUN "〈ファイル名〉"，R

R型のRUNコマンドは，やはりディスクからプログラムをロードして，すぐにそのプログラムの実行を開始します．しかしこのとき，直前までOPENされていたデータファイルはCLOSEされません．ただし，記憶装置内のデータやプログラムはすべて消去されます．

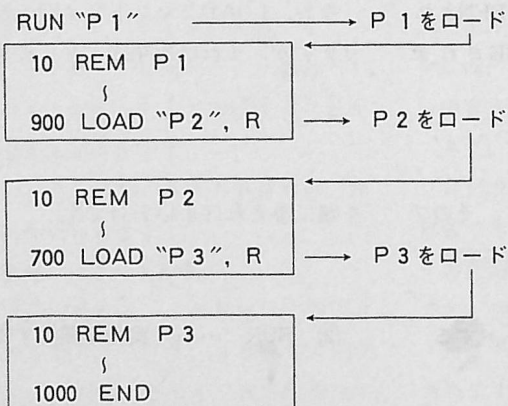
また次のものは，R型のLOADコマンドです．これの働きはR型のRUNコマンドと同一です．

LOAD "〈ファイル名〉"，R

R型のRUNコマンドやR型のLOADコマンドは，通常は必要ありませんが，プログラムのチェインを行なうときなどに利用されます．

プログラムが非常に大きくて記憶装置に入りきれない場合には，そのプログラムをいくつかの部分に分割して，それぞれの部分に別のファイル名を付けてディスクにセイブしておき，これを順次ロードして実行するという方法があります．これをプログラムのチェインといいます．

例えば，非常に長いプログラムを3つの部分に分割して，それぞれP1，P2，P3というファイル名でディスクにセイブしておきます．これを実行するときは，まずプログラムP1をロードしてその実行を開始させます．プログラムP1およびP2には，最後の命令として，R型のRUNまたはR型のLOADを書いておきます．すると，P1の実行の最後の所でP2がロードされ実行されます．P2の最後の所でP3がロードされ実行されます．

これがプログラムのチェインです．次図を見てください．

プログラムのチェインを行なう場合は，次の点に注意が必要です．

あるプログラム（例えばＰ２）をロードすると，その前のプログラム（例えばＰ１）そのものおよびそこで使用していた変数はすべて消去されます．

したがって，各プログラム(Ｐ１，Ｐ２，Ｐ３)の相互間でGOTOやGOSUBなどを用いることはできません．例えばＰ２の中からＰ１の中へGOTOで行くことはできません．

また，各プログラム間でのデータの受け渡しは，記憶装置内では（ロードのとき消去されるから）できません．そこで，データの受け渡しはディスクのデータファイルを介して行なうようにします．

11·3　シーケンシャルファイルの使い方

ディスクにおけるデータファイルには，シーケンシャルファイルとランダムファイルとがあり，それぞれ特徴を持っています．シーケンシャルファイルは順次的にしか読み書きできないという短所を持っていますが，プログラムの書き方はランダムファイルより簡単です．

本節では，シーケンシャルファイルを使う場合の各種の命令を紹介し，読み書きの方法を説明します．

11·3·1　シーケンシャルファイルのあらまし

シーケンシャルファイルとは，データの読み書きが順次的に行なわれるファイルです．第10章で述べたテープファイルも典型的なシーケンシャルファイルのひとつです．

例えば次のように６個のデータを並べて記録したシーケンシャルファイルがあるとします．

TOKYO	SHINYOKOHAMA	ODAWARA	ATAMI	MISHIMA	SHIZUOKA

このファイルのデータを読むときは，先頭のデータから順に読むことしかできません．３番目の「ODAWARA」というデータを読みたい場合でも，いきなり「３番目のデータを読め」というように命令することはできません．「TOKYO」，「SHINYOKOHAMA」というように先頭から順次に読んで行かないと，「ODAWARA」を読むことはできません．

また，５番目の「MISHIMA」を「NUMAZU」に書き変えることもできません．「MISHIMA」と「SHIZUOKA」の間に「NUMAZU」を挿入することもできません．このように，シーケンシャルファイルでは，ファイルの一部分だけを書き変えたり，間にデータを挿入したりすることはできません．

しかし，最後のデータの後にさらにデータを追加して記録することはできます．例えば「SHIZUOKA」の後に「HAMAMATSU」を追加することはできます．

以上のことをまとめると，シーケンシャルファイルに対してできることは次の３つになります．すなわち３つのモードのうちのいずれかでファイルを取扱うことになります．

① データを書き込んで，新規にファイルを作成する．………………OUTPUTモード
② 既存のファイルを順次に読む．…………………………………………INPUTモード

③　既存のファイルの末尾に，データを追加して書き込む．…………APPENDモード

シーケンシャルファイルを取扱うときには，次のようなステートメントや関数が用いられます．

OPEN文，CLOSE文，
PRINT#文，PRINT# USING文，
INPUT#文，LINEINPUT#文，
EOF関数

これらについて，ひととおり説明を加えておきます．

OPEN "<ファイル名>" FOR <モード> AS #<ファイル番号>

これはOPEN文です．OPEN文は，ファイルを使用することをコンピュータに知らせるために用いるステートメントです．つまりファイル使用開始の宣言です．ファイルの使用を開始するときは，必ずOPEN文を実行させることが必要です．これによって，そのファイルが使用（読み書き）可能な状態になります．OPEN文を実行させることを，ファイルを「OPENする」とか「開く」といいます．

モードとしては，先に述べた3種類（OUTPUT，INPUT，APPEND）のうちのいずれかを指定します．

OPENされている各ファイルに対して，読み書きすべき現在位置を，コンピュータは常に記憶しています．この位置の記憶を「ファイルポインタ」といいます．

OPEN文でOUTPUTモードを指定すると，新しいファイルを設定して，そのファイル名をディスクに登録します．そしてファイルポインタをそのファイルの先頭の位置にセットします．このとき，OPEN文で指定されたファイル名がもしすでにディスク内に存在していたならば，その既存のファイルの先頭にファイルポインタがセットされます．この場合は，その既存のファイルの内容は壊されます．

OPEN文でINPUTモードを指定すると，ファイルを見付けてその先頭にファイルポインタをセットします．

OPEN文でAPPENDモードを指定すると，ファイルを見付けてそのファイルの最後のレコードの直後の位置にファイルポインタをセットします．

INPUTモードあるいはAPPENDモードが指定されたとき，もしファイルがディスク上に存在していなければ，エラー（File not found）となります．

OPEN文のモード指定を省略すると，もしファイルがディスク上にあればその先頭にファイルポインタをセットし，またもしファイルが存在していなければ，新しいファイルを設定してその先頭にファイルポインタをセットします．

次に，OPEN文で指定するファイル番号ですが，これは，そのファイルがOPENされている間だけ通用する番号であり，そのファイルを指定するために，PRINT#文やINPUT#文その他の中で使われます．

ある時点において，同時にOPENされている各ファイルに対しては，それぞれ異なったファイル番号を割当てなければなりません．同時にOPENされるファイルの個数の限界は，11·1·2で述べたようにディスクシステムプログラムのスタート時において，「How many files?」の問いに対してKey-inしてあります．ファイル番号は，このKey-inした数より大きく指定することはできません．

以上のことをまとめると，OPEN文の機能は次のように要約されます．

① 使用を開始するファイル名をコンピュータに知らせること．

② モード指定によって，ファイルポインタの初期位置をセットすること．

③ ファイル番号を割当てること．

〔例1〕 OPEN "ABC" FOR OUTPUT AS #1

CLOSE #〈ファイル番号〉, #〈ファイル番号〉, ………

これはCLOSE文です．

〔例2〕 ① CLOSE #1

② CLOSE #1, #2

③ CLOSE

この例③ではファイル番号が指定されていませんが，この場合は，OPENされているファイルをすべてCLOSEします．なお，ファイル番号の頭に付ける記号（#）は省略してもよろしい．

CLOSE文はOPEN文に対応するものであり，ファイル使用終了の宣言です．CLOSE文が実行されると，OPEN文で指定されたファイル名とファイル番号との対応関係は解消します．したがって，そのファイル番号を別のファイルに割当ててOPENすることができるようになります．

CLOSE文を実行させることを，ファイルを「CLOSEする」とか「閉じる」といいます．

CLOSEされたファイルを再びOPENしてもかまいません．そのときに割当てるファイル番号は，前回のファイル番号と同じでも異なっていてもかまいません．

PC－8001の記憶装置とディスクとの間のデータの入出力は，直通で行なわれるのではなく，中継場所を経由して行なわれます．この中継場所は「バッファ」と呼ばれ，1セクタ分の広さがあります．

ディスクシステムのスタート時点において「How many files?」の問いに利用者が応答した数（すなわち，同時にOPENされるファイルの個数の限界）だけのバッファを，コンピュータは用意します．ファイルがOPENされると，そのファイルに対して1個ずつのバッファが割当てられます．どのバッファを割当てたかは，ファイル番号によって識別されます．

ファイルはOPENされている間は，自分に割当てられたバッファを専有して使います．そのファイルがCLOSEされると，専有していたバッファはそのファイルから解放され，他のファイルに割当てることが可能になります．このように，バッファを専有から解放することは，CLOSE文の重要な働きです．これはまた，ファイル名からファイル番号を切り離すということに相当します．

OPENしたファイルは必ずCLOSEすることが原則ですが，CLOSE文を実行しなくても，次の場合には自動的にCLOSEした状態になります．

{ END, NEW, REMOVEなどを実行したとき，
  RUNやLOAD（R型でないもの）を実行したとき．

なお，STOP文はCLOSEの機能を持っていません．

PRINT# 〈ファイル番号〉, 〈出力並び〉

シーケンシャルファイルにデータを書くときに，このPRINT#文を用います．PRINT#文は，PRINT文でディスプレイ画面に表示するときと同じ形で，出力並びの内容をディスクに書きます．

ファイル番号は，OPEN文で指定したものと同じ番号を書きます．

出力並びには，数値データや文字データを，変数名，定数あるいは式の形で書くことができます．文字データについては，その区切り方に注意しないと，そのデータを読むときに悪い影響を及ぼすことがあります．それを防ぐには次のようにすればよろしい．

文字データを区切るために，引用符で囲まれたコンマ（"，"）を用います．

例えば，PRINT＃１，A$；"，"；B$

また，文字データ自身の中にコンマや意味のある先頭の空白などが含まれている場合は，CHR＄(34) でその文字データを囲みます．

例えば，PRINT＃１，CHR$(34)；"ABC, DEF"；CHR$(34)

PRINT＃　〈ファイル番号〉，USING "〈書式仕様〉"；〈出力並び〉

このPRINT＃ USING文も，シーケンシャルファイルにデータを書くときに用いられます．その書き方は，第９章（9.2.2）で示したものと同じ要領です．

INPUT＃　〈ファイル番号〉，〈入力並び〉

このINPUT＃文は,シーケンシャルファイルに書かれているデータを読むときに用いられます．入力並びとは，ディスクから読んだデータを記憶すべき場所を指定するものであり，変数名で表わします．変数名が複数個のときはコンマ（，）で区切ります．

LINE INPUT＃　〈ファイル番号〉，　〈文字型変数名〉

このLINE INPUT＃文も，シーケンシャルファイルに書かれているデータを読むときに用いられます．キーボードから入力するときのLINE INPUT文と同じ要領で，ディスクからデータを読みます．すなわち，コンマや空白などもデータの構成要素とみなし，途中を区切らずに１個の文字データとして読んで，指定された文字変数にそれを入力します．

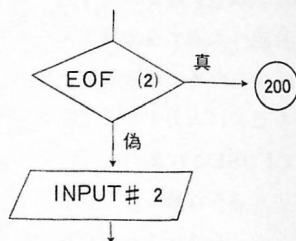
EOF　(〈ファイル番号〉)

これはEOF関数です．この関数の名前（EOF）は「End Of File」を縮めて付けられたものです．

シーケンシャルファイルに記録されているデータを読むときは，ファイルの先頭のデータから順に読んで行きます．そして，ファイルの最後のデータを読んだときには，EOF関数の値は－１（真）となります．そうでなければ（ファイル内にまだデータが残っていれば），EOF関数の値は０（偽）となります．

EOF関数は，データの入力（INPUT＃）を行なうときに，ファイルの終りを検出するために使われます．

〔例 3〕

```
      ⁝
100  IF  EOF(2)  GOTO  200
110  INPUT＃2, A
      ⁝
```

この例では，ファイル番号2のファイルについて，ファイルの終りの検出を行なっています．

11·3·2 シーケンシャルファイルの新規作成

シーケンシャルファイルを新規に作成する（ファイルを設定して，ディスクにデータを書き込む）ときの一般的な手順の概要は次のとおりです．

① ファイルをOPENする．………OPEN文(OUTPUTモード)
② データを書く．………PRINT＃文あるいはPRINT＃ USING文
③ ファイルをCLOSEする．………CLOSE文

次に示す例は，シーケンシャルファイル新規作成の定型的なものです．

〔例〕

```
100  REM  Sequential  File  Output
110  OPEN  "DATA01"  FOR  OUTPUT  AS  #1………ファイルを開く
120  INPUT  "CODE＝"; C$  ………データをキーボードから入力
130  IF  C$＝"XXX"  GOTO  180  ………入力データの終りの判定
140  INPUT  "NAME＝"; N$  ………}
                                  } データをキーボードから入力
150  INPUT  "TANKA＝"; T  ………}
160  PRINT＃1, C$; ","; N$; ","; T  ………ディスクにデータを書く
170  GOTO  120
180  CLOSE  ＃1  ………ファイルを閉じる
```

このプログラムでは，何品目かの商品について，商品コード（C＄），商品名（N＄），単価（T）をキーボードから入力して，それをディスクに書き込みます．

プログラムの骨組みの要点は，110番のOPEN文，160番のPRINT＃文，180番のCLOSE文です．ファイル名は「DATA01」であり，ファイル番号は1番としてOPENしています．シーケンシャルファイルの新規作成の場合はOUTPUTモードを指定します．

PRINT＃文やCLOSE文で用いているファイル番号（1番）は，OPEN文で指定したものと同じ番号です．

PRINT＃文の出力並びでは，引用符で囲まれたコンマ（","）を用いて，データを区切っています．

このプログラムの120番から150番までの行は，ディスクに書き込むべきデータをキーボードから記憶装置へ入力する部分です．入力データの終りを判定するために「XXX」を用いています．XXXをC＄に入力すれば180番へ飛んで，ファイルはCLOSEされます．これでディスクへのデータの書き込みは終了し，DATA01という名前のシーケンシャルファイルが新規に作成されたことになります．

この例では，データをキーボードから入力して，それをそのままディスクに書き込んでいます．しかし，このことは，ディスクファイルにとって本質的なことではありません．どこから入力したデータであろうが，あるいは計算処理の結果できたデータであろうが，要するに，記憶装置内にあるデータならばそれをディスクに書き込むことができます．

11·3·3　シーケンシャルファイルの読み込み

すでに作成されて存在しているシーケンシャルファイルのデータを読んで，それを記憶装置へ入れるときの一般的な手順の概要は次のとおりです．

① ファイルをOPENする．………OPEN文（INPUTモード）
② データを読む．………INPUT＃文あるいはLINEINPUT文
③ ファイルをCLOSEする．………CLOSE文

なおこの他に，EOFの検出を組込みます．

次に示す例は，シーケンシャルファイルの読み込みの定型的なものです．

〔例〕

```
200 REM  Sequential  File Input
210 OPEN "DATA01" FOR  INPUT  AS#1…ファイルを開く
220 IF  EOF(1) GOTO  260 ………ファイルの終り（EOF)の検出
230 INPUT#1, A$, B$, C ………ディスクからデータを読む
240 PRINT  A$, B$, C ………読んだデータを画面に表示する
250 GOTO  220
260 CLOSE  #1………ファイルを閉じる
```

このプログラムは，前項（11·3·2）のプログラムで作成されたファイルのデータを読んで，それを記憶装置内の変数に入れるものです．

プログラムの骨組みの要点は，210番のOPEN文，230番のINPUT＃文，260番のCLOSE文およびEOF検出のための220番のIF文です．240番のPRINT文は，ディスクファイルにとって直接の関係はなく，ひとつの例に過ぎません．ディスクから読んだデータに対して任意の処理をすることができます．

ファイルを読むときにOPEN文で指定するファイル名は，そのファイルを作成するときに付けたファイル名と同一でなければなりません．この例では「DATA01」です．

シーケルシャルファイルを読むときは，INPUTモードを指定します．

INPUT＃文やCLOSE文あるいはEOF関数で用いているファイル番号（1番）は，OPEN文で指定したものと同じ番号です．

前項のプログラムでファイルに書き込むとき，PRINT＃文の出力並びではC＄，N＄，Tという変数名を用いました．ところが本項のプログラム

のINPUT＃文では，A＄，B＄，Cという変数名を用いています．このように，書くときと読むときとでは，異なった変数名を用いてもよいのです．ただし，その型は一致していなければなりません．

シーケンシャルファイルを読むときに，EOFの検出をしないでどんどん読ませると，やがてデータがなくなったにもかかわらずINPUT#文を実行させる結果になります．その場合にはエラー（Input past end）となります．

11·3·4　シーケンシャルファイルへのデータの追加

すでに作成されて存在しているシーケンシャルファイルの末尾に，さらにデータを追加して書き込むことができます．その一般的な手順の概要は次のとおりです．

① ファイルをOPENする．………OPEN文（APPENDモード）
② データを書く．
③ ファイルをCLOSEする．
（②③：ファイルの新規作成の場合と同じ）

この手順はファイルの新規作成の場合とほとんど同じです．異なるのは，OPEN文のモード指定だけです．シーケンシャルファイルへのデータの追加の場合には，APPENDモードを指定します．そうすると，ファイルの末尾のデータの直後にファイルポインタがセットされます．この点が他のモードと異なります．

次に示す例は，シーケンシャルファイルへのデータの追加の定型的なものです．

〔例〕

```
300 REM  Sequential File Append
310 OPEN "DATA01" FOR APPEND AS #1 ………APPENDモードを指定
320 INPUT "CODE="; C$
330 IF C$= "XXX" GOTO 380
340 INPUT "NAME="; N$
350 INPUT "TANKA="; T
360 PRINT# 1, C$; ","; N$; ","; T
370 GOTO 320
380 CLOSE #1
```

このプログラムは，前々項（11·3·2）のプログラムで作成されたファイルの末尾に，データを追加して書き込むものです．

310番のOPEN文でAPPENDモードを指定しています．320番以降は，ファイルの新規作成の場合と同じ内容です．

このようにしてデータを追加されたファイルを読むときは，前項（11·3·3）のプログラムがそのまま使えます．

11·4　ランダムファイルの使い方

ランダムファイルは，シーケンシャルファイルにはない利点を持っています．仕事の性格に合わ

せて使えば，効果的なデータ処理ができます．そのかわり，プログラムの書き方はシーケンシャルファイルよりもやや複雑になります．しかしその定型を理解すれば，誰でも容易に利用することができます．

本節では，ランダムファイルを使う場合の各種の命令を紹介し，読み書きの方法を説明します．

11·4·1　ランダムファイルのあらまし

シーケンシャルファイルでは，読み書きは順次的にしかできません．それに比べてランダムファイルでは，ファイル内の任意の場所を指定して，他の場所には触れずに直接に読み書きすることが可能です．

例えば，前節（11·3·1）の始めに示した6個のデータについていえば，ランダムファイルの場合は，「3番目のデータを読め」と命令すれば，1番目のデータ（TOKYO)や2番目のデータ(SHIN-YOKOHAMA)を読まずに，直接に「ODAWARA」を読むことができます．

また，「5番目の場所に書け」と命令すれば，他の場所には触れずに，直接に書き込みます．ということは，「MISHIMA」を「NUMAZU」に書き変えることができるということです．ファイルへのデータの追加も，もちろん可能です．

ランダムファイルを取扱うときには，次のようなステートメントや関数が用いられます．

OPEN文，CLOSE文，
PUT文，GET文，
FIELD文，LSET文，RSET文，
MKI$，M KS$，MKD$関数，
CVI，CVS，CVD関数，
LOF関数，LOC関数

これらについて，ひととおり説明を加えておきます．

OPEN"<ファイル名>" AS#<ファイル番号>

これはランダムファイルに対するOPEN文です．ファイル名やファイル番号についてはシーケンシャルファイルの場合と同じです．

モードは指定しません．すなわち，一度OPENしたならば，そのランダムファイルをCLOSEするまでの間に，書くことも読むことも，データを追加することもできるのです．

前節（11·3·1）で述べたように，OPEN文によってファイルにバッファが割当てられます．ランダムファイルに割当てられたバッファのことを「ランダムバッファ」と呼びます．ランダムバッファはデータの読み書きの中継場所として重要な役割を果します．

OPENしたファイルは使用後にCLOSEしなければなりませんが，ランダムファイルに対するCLOSE文の書き方は，シーケンシャルファイルの場合とまったく同じです．

```
PUT #〈ファイル番号〉，〈レコード番号〉
```

```
GET #〈ファイル番号〉，〈レコード番号〉
```

PUT文は，ランダムバッファにある1レコード分のデータを，ディスク（ランダムファイル内の，レコード番号で指定された位置）に書き込みます．

GET文の動作はPUT文の逆であり，レコード番号で指定された1レコード分のデータを，ディスクから読んでランダムバッファに入れます．

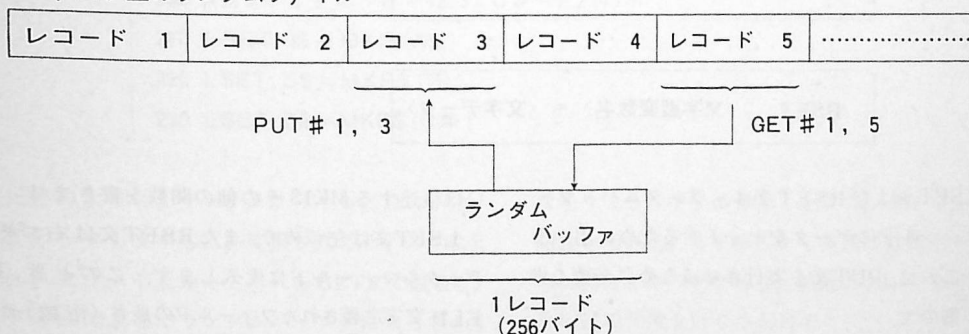
さて，1回に入力あるいは出力される1組のデータを「レコード」と呼びます．ランダムファイルへの入出力においては，1レコードの長さを常に一定に固定して取扱います．その長さは256バイト，すなわち1セクタにきめられています．ランダムバッファは，この1レコード分を記憶できる広さ（256バイト）を持っています．

PUT文やGET文は，ランダムバッファとディスクとの間で，1レコード単位でデータのやり取りを行なうのです．そして，ランダムファイルの中には，一定（256バイト）の長さのレコードを記録する場所が並んでいます．このそれぞれの記録場所に付けた番号が「レコード番号」です．

次のように，レコード番号を省略したPUT文，GET文の形もあります．この場合は，その直前にPUTまたはGETしたレコードの次のレコード番号が自動的に指定されます．

```
PUT #〈ファイル番号〉
```

```
GET #〈ファイル番号〉
```

次図はランダムファイルとランダムバッファとの関連を示すものです．

PUT文はランダムバッファの内容をディスクに書くのですから，PUT文を実行させる前に，ディスクに書かれるべき1レコード分のデータをバッファの中にセットしなければなりません．そのためには，バッファ内のデータ領城を定義しておくことが必要です．またGET文の場合も，ディスク

から読まれた1レコード分のデータを受け入れる準備として，やはりデータ領域の定義をしておくことが必要です．

ランダムバッファ内の個々のデータ領域を「フィールド」と呼びます．このフィールドの構成を定義するために，次のFIELD文を用います．

FIELD＃〈ファイル番号〉，〈フィールドの指定〉

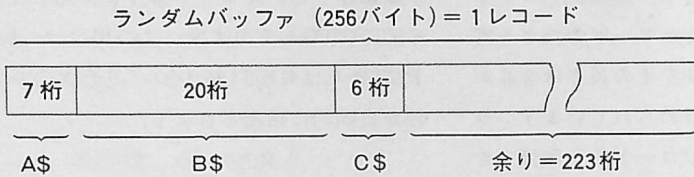
FIELD文の中の「フィールドの指定」は次の形に書きます．

〈長さ〉AS〈文字型変数名〉，〈長さ〉AS〈文字型変数名〉，…………………

〔例1〕 FIELD ＃1，7 AS A$，20 AS B$，6 AS C$

この例では，ランダムバッファ内は次図のように領域が定義されます．

ランダムバッファ (256バイト)＝1レコード

7桁	20桁	6桁	
A$	B$	C$	余り＝223桁

この例で見られるように，1レコードの中のデータが少ないときは，余りの領域が大きくなり，ディスクの記録場所をむだ使いすることになります．しかしこれは，高速なランダムアクセスを行なうために，ある程度やむを得ぬことです．

なお，フィールドの指定の中で用いた変数には，次に述べるLSETあるいはRSET以外の命令(INPUT文，READ文，代入文など）によって値を入れてはいけません．

LSET 〈文字型変数名〉＝〈文字データ〉

RSET 〈文字型変数名〉＝〈文字データ〉

LSETおよびRSET文は，ランダムバッファのフィールドにデータをセットするために用います．これは，PUT文を実行させるために必要な準備作業です．

LSET文やRSET文に書く文字型変数名は，FIELD文のフィールド指定において定義されていなくてはなりません．また，＝の右の文字データとしては，文字型の変数名，文字定数，文字式あるいは後述するMKI$その他の関数を書きます．

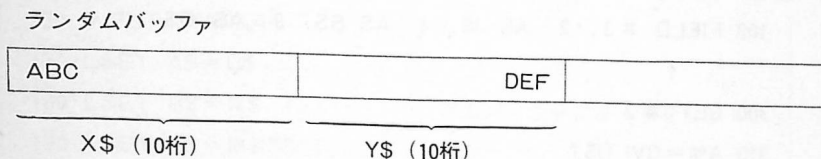
LSET文は左づめで，またRSET文は右づめで，データをフィールドに代入します．このとき，FIELD文で定義されたフィールドの長さ（桁数）がデータの長さより大きければ，余った桁には空白がつめられます．逆にデータの方がフィールドよりも長い場合には，フィールドに入りきれない部分は無視されます．

〔例２〕

```
100 FIELD ＃2, 10 AS X$, 10 AS Y$, ………
    ⁝
200 LSET X$＝"ABC"
210 RSET Y$＝"DEF"
```

この例では，次図のようにランダムバッファにデータがセットされます．

ランダムバッファ

ABC	DEF	
X$（10桁）	Y$（10桁）	

PUT文やGET文による読み書きにおいては，すべてのデータを文字の形で取扱います．したがって，数値データをランダムファイルに書き込むときは，それを文字形式に変換してランダムバッファのフィールドにセットしなければなりません．この変換のために，下記の３種の関数を用います．

MKI$（<整数型データ>）………整数型の数値を２バイトの文字列に変換する

MKS$（<単精度型データ>）………単精度型の数値を４バイトの文字列に変換する

MKD$（<倍精度型データ>）………倍精度型の数値を８バイトの文字列に変換する

〔例３〕

```
100 FIELD＃3, 2 AS I$, 4 AS S$, 8 AS D$
    ⁝
200 A%＝1234 :B＝12.3:C＃＝1.2345＃
210 LSET I$＝MKI$(A%)
220 LSET S$＝MKS$(B)
230 LSET D$＝MKD$(C＃)
```

上述のように，ランダムファイルにおいては，数値データも文字形式に変換されてディスクに記録されます．したがってそれをGET文で読むと，文字形式のままでランダムバッファに入ってきます．そのままではそのデータを演算その他の処理に使うことはできません．そこで，下記の３種の関数を用いて，文字形式から普通の数値データの形に戻します．すなわち，前述のMKI$などの関数の逆の変換を行なうわけです．

CVI（<文字型変数名>）………２バイトの文字列を整数型の数値に変換する

CVS（〈文字型変数名〉）

………4バイトの文字列を単精度型の数値に変換する

CVD（〈文字型変数名〉）

………8バイトの文字列を倍精度型の数値に変換する

上記の文字型変数名はいずれも，FIELD文のフィールド指定で定義されたものです．

〔例4〕

```
100 FIELD ＃3， 2  AS I$， 4  AS S$， 8  AS D$
        ∫
300 GET ＃3
310 A%＝CVI（I$）
320 B＝CVS（S$）
330 C＃＝CVD（D$）
```

LOF （〈ファイル番号〉

これはLOF関数です．ランダムファイルにそれまでに書き込まれた最大のレコード番号が，この関数の値として与えられます．したがって，ランダムファイルの末尾にレコードを追加して書き込むときは，PUT文のレコード番号として，下記の値を指定すればよいことになります．

LOF （〈ファイル番号〉）＋1

ただし次のことに注意しなければなりません．LOF関数の値は，そのファイルをOPENした時点で与えられ，その後CLOSEするまでは，関数の値は更新されないのです．

LOC （〈ファイル番号〉）

これはLOC関数です．直前にPUTまたはGETしたレコードの次のレコード番号が，この関数の値として与えられます．すなわち，レコード番号の指定を省略したPUT文またはGET文が読むべきレコードの番号ということになります．

また，シーケンシャルファイルに対して LOC関数を用いると，OPENしてから現在までに読み書きしたセクタの総数が与えられます．

11·4·2　ランダムファイルの新規作成

ランダムファイルを新規に設定して，ディスクにデータを書き込むプログラムの例を，次に掲げます．

〔例〕

```
100 OPEN "RNDF1" AS  #1
110 FIELD #1, 7 AS A$, 20 AS B$, 4 AS C$
120 X= 0
130 INPUT "CODE="; D$          ┐
140 IF D$= "end" GOTO 230      │
150 INPUT "NAME="; N$          │キーボードからのデータの入力
160 INPUT "TANKA="; T          ┘
170 LSET A$=D$                 ┐
180 LSET B$=N$                 │ランダムバッファにデータをセット
190 LSET C$=MKS$(T)            ┘
200 X=X+1
210 PUT #1, X ……バッファからディスクへ1レコードを転送
220 GOTO 130
230 CLOSE #1
```

このプログラムでは，レコード番号を変数Xで指定しています．Xの初期値を0にしておいて，それを1ずつ増加させてPUT文で用いています．したがって，キーボードから入力した順に，ファイルのレコードが並ぶことになります．

11·4·3　ランダムファイルへのレコードの追加

ランダムファイルの末尾にさらにレコードを追加して書き込むには，LOF関数を利用します．

例えば，前項（11·4·2）に掲げたプログラムの中の120番の行のみを，下記のように書き変えれば，レコードの追加を行なうことができるようになります．

```
120 X =LOF (1)
```

LOF関数は，ファイルをOPENしたときに，そのファイルの末尾にあるレコードの番号を与えてくれます．したがってこのプログラムによって，レコードの追加ができるわけです．LOF関数については11·4·1で紹介してあります．

11·4·4　ランダムファイルの読み込み

ランダムファイルに書き込まれているデータをコンピュータの中に読み込むプログラムの例を，次に掲げます．

〔例〕
```
100 OPEN "RNDF 1" AS #1
110 FIELD #1, 7 AS A$, 20 AS B$, 4 AS C$
120 INPUT "RECORD-NO="; X ………レコード番号をキーボードから入力
130 IF X=0 GOTO 190
140 GET #1, X ……………ディスクからバッファへ1レコードを転送
150 PRINT A$     ┐
160 PRINT B$     ├読んだデータを画面に表示
170 PRINT CVS (C$) ┘
180 GOTO 120
190 CLOSE #1
```

このプログラムでは，読みたいレコードの番号を，120番のINPUT文でキーボードから入力しています．このような方法では，どのデータが何番のレコードに記録されているかということを，人間の側で覚えておかねばならないことになります．現実問題としては，これでは不便です．

人間の側でキー入力するものとしては，通常は商品コード，従業員コードなどが用いられるでしょう．そしてそれらをファイルのレコード番号に変換する作業は，プログラムによってコンピュータに行なわせるべきものです．

このためには，商品コードなどとレコード番号との対応表が必要になります．そのような方法は，第13章の応用プログラム例の中で紹介することにします．

11·4·5　ランダムファイルのレコードの書き変え

ランダムファイルの中のあるレコードを指定して，その記録内容を書き変えることができます．そのプログラム例を次に掲げます．

〔例〕
```
100 OPEN "RNDF 1" AS #1
110 FIELD #1, 7 AS A$, 20 AS B$, 4 AS C$
120 INPUT "RECORD-NO="; X ………レコード番号をキーボードから入力
130 IF X=0 GOTO 220
140 INPUT "CODE="; D$   ┐
150 INPUT "NAME="; N$   ├キーボードからデータ(新)を入力
160 INPUT "TANKA="; T   ┘
```

```
170 LSET A$＝D$
180 LSET B$＝N$         ランダムバッファにデータをセット
190 LSET C$＝MKS$(T)
200 PUT ＃1, X ………バッファからディスクへ1レコードを転送
210 GOTO 120
220 CLOSE ＃1
```

このプログラムでも，前項（11･4･4）と同様に，レコード番号をキーボードから入力して指定しています．したがって，前項と同様な不便さがあります．これについてはやはり第13章を参照してください．

ランダムファイルの新規作成，レコードの追加，読み込み，書き変えなどについて，今までの各項で別々にプログラム例を掲げてきましたが，実際には，これらの処理を組合わせたプログラムを作って用いることになります．

11･5　ファイル名なしの読み書き

前節までに述べてきたディスクへの読み書きの方法は，すべてファイル名を用いて，ディレクトリやFATの管理のもとで行なうものでした．本節で紹介するのは，ファイル名を用いずに，直接にドライブ番号，トラック番号，セクタ番号を指定して読み書きする方法です．

この方法によるディスクへの読み書きは，ディレクトリやFATの管理を受けません．したがって先に述べたMOUNTコマンドを用いる必要がありません．しかし，うっかり命令の仕方を誤ると，ディスク上の記録を破壊するおそれがあるので，注意しなければなりません．

この方法は，DSKI＄という関数と，DSKO＄というステートメントを用いて行ないます．また，この2つの命令に専用のバッファがあります．そのバッファの番号は0番です．前節までに述べたシーケンシャルファイルやランダムファイルで用いるバッファの番号は，ファイル番号と同じで1番から15番までの範囲です．

次のものはDSKI＄関数です．

DSKO$ 〈ドライブ番号〉，〈トラック番号〉，〈セクタ番号〉

これはステートメントではなく関数ですから，例えば次のように代入文の右辺で用います．

Y$＝DSKI$ （2，3，4）

この代入文を実行すると，第2ドライブのディスクの第3トラックの第4セクタに記録されている1レコード（256文字）が読まれて，0番のバッファに入り，さらにそれが変数Y＄に入ります．

ここでひとつの問題が生じます．それは，文字型変数（この例ではY＄）には255文字の文字列しか記憶できないということです．したがって，ディスクの1レコード（256文字）の中の末尾の1文字はY＄に入りきれないで失われます．

このようなことを防ぐには，0番のバッファの中を，FIELD文によっていくつかのフィールドに分割しておけばよいのです．次の例を見てください．

〔例１〕

```
100 FIELD ＃0, 7 AS A$, 20 AS B$, 4 AS C$
110 INPUT "DRIVE"; D         ┐
120 INPUT "TRACK"; T         ├ドライブ，トラック，セクタの番号を入力
130 INPUT "SECTOR"; S        ┘
140 DM$＝DSKI$ (D, T, S) ………DSKI$を動作させる
150 PRINT A$                 ┐
160 PRINT B$                 ├読んだデータを画面に表示
170 PRINT CVS (C$)           ┘
```

この例の140番の代入文を実行すると，ディスクが読まれます．左辺の変数（DM＄）には255文字が入ります．ここではDM＄はダミーとして用いています．

次のDSKO＄はステートメントです．

```
DSKI$ (<ドライブ番号>, <トラック番号>, <セクタ番号>)
```

このステートメントを実行すると，番号で指定したディスク上のセクタに，0番のバッファの内容（1レコード）を書き込みます．したがってその前に，0番のバッファにデータをセットしておくことが必要です．次の例を見てください．

〔例２〕

```
100 FIELD ＃0, 7 AS A$, 20 AS B$, 4 AS C$
110 INPUT "CODE＝": D$        ┐
120 INPUT "NAME＝"; N$        ├データをキーボードから入力
130 INPUT "TANKA＝"; T        ┘
140 LSET A$＝D$               ┐
150 LSET B$＝N$               ├0番のバッファにデータをセット
160 LSET C$＝MKS$ (T)         ┘
170 INPUT "DRIVE, TRACK, SECTOR"; D, T, S
180 DSKO$ D, T, S ………ディスクに書き込む
```

11·6　ディスクに関するその他の命令

前節までに述べなかったディスク関係の命令をいくつか本節で紹介しておきます．

(1)　ファイル名のリスト

ディスクに記録されているファイルの名前を知りたいときは，次のコマンドを用います．

```
FILES <ドライブ番号>   ……………ディスプレイ画面に表示する
```

LFILES 〈ドライブ番号〉

……………プリンタで用紙に印字する

これらのコマンドを実行すると，そのディスクに含まれているファイルの名前と大きさが出力されます．

なお，ドライブ番号の指定を省略すると，1番のドライブが指定されたものとみなされます．

〔例1〕 FILES 2

(2) ファイル名の付けかえ

すでに存在しているファイルのファイル名は，次のコマンドにより，別の名前に変更することができます．このコマンドを実行した後は，そのファイルは，同じディスク上の同じ場所に，新しい名前で存在します．

NAME "〈旧ファイル名〉" AS "〈新ファイル名〉"

〔例2〕 NAME "ABC" AS "DEF"

(3) ファイルの削除

すでに存在しているファイルは，次のコマンドによってディスクから削除することができます．

KILL "〈ファイル名〉"

〔例3〕 KILL "DATA 01"

(4) 属性の指定（ファイルの保護）

ディスクに書かれているデータを誤って消してしまったり，みだりにファイル内容を変更してしまったりすると，大きな損害となります．このような事態を防ぐために，またディスク記録の信頼性を高めるために，属性を指定するという方法があります．

属性には次の2種があります．

① 書き込み禁止
② Read after Write（書き込み状態を検査する）

一度指定した属性を解除することもできます．属性を指定したり解除したりするにはSETというコマンドを用います．その中で，次のような属性文字を使います．

① 属性文字P（大文字）……………書き込み禁止を指定するとき
② 属性文字R（大文字）……………Read after Writeを指定するとき
③ 上記以外の文字…………………属性指定を解除するとき

SETというコマンドには下記のように3種の形式があります.

SET "〈ファイル名〉", "〈属性文字〉"

〔例4〕 SET "ABC", "P"

この例では, ディスク上に存在するABCという名前のファイルに対して, 書き込み禁止属性を指定しています. これが指定されると, このファイルに書き込むことはできなくなります.

〔例5〕 SET "ABC",""

この例は,ABCという名前のファイルに指定されていた属性を解除するものです.

SET 〈ドライブ番号〉, "〈属性文字〉"

〔例6〕 SET #1, "R"

この形式で指定された属性は, そのファイルがOPENされている間だけ有効です.

SET# 〈ファイル番号〉, "〈属性文字〉"

〔例7〕 SET 1, "P"

この例では, 1番のドライブに入っている1枚のディスクに対して, P属性を指定しています.

理解度テスト

(1) ディスクに関する次の文章のうち, 内容の正しいものには○をつけ, また誤りのあるものには×をつけなさい.

① ファイル番号には, 1～32767の範囲の数を用いる.
② ディスクに記録されているデータをディスプレイ画面に表示するときは, PRINT#文を用いる.
③ INPUT#文は, シーケンシャルファイルを読むときに用いられる.
④ シーケンシャルファイルへデータを追加するときは, APPENDモードで行なう.
⑤ ランダムファイルはOPENせずに読み書きするのが原則である.
⑥ EOF関数の引数は, ファイル番号である.
⑦ PUT文はランダムファイルにデータを書くときに用いる.
⑧ GET文の中には, ドライブ番号とファイル名を書く.
⑨ ランダムバッファの大きさは1クラスタ分である.
⑩ ランダムファイルをOPENするときは, モードを指定する必要はない.
⑪ LSET文を実行すると, バッファの内容がディスクに書き込まれる.
⑫ EOF関数は, シーケンシャルファイルの終りを検出するときに用いられる.
⑬ FIELD文の中に書く変数名は, 必ず文字型でなければならない.

⑭　MKI＄関数は，直接にセクタ番号を指定してディスクのデータを読むために用いる．

⑮　CVI関数は，２バイトの文字列を整数型に変換してディスクに書き込む働きをする．

(2)　次のことを行なうための命令を，BASIC言語で書きなさい．

①　ドライブ２番にあるディスクに記録されているファイルの名前を，画面に表示する．

②　ABCという名前のファイルを，ディスク上から削除する．

③　DATAという名前のファイルに，書き込み禁止属性を指定する．

④　KEISANという名前のプログラムを，１番のドライブのディスクに，バイナリ形式でセイブする．

⑤　２番のドライブにセットしたディスクの使用開始を宣言する．

⑥　KEISANという名前のプログラムを，１番のドライブのディスクから記憶装置に読み込んで，実行を開始する．

⑦　DATA１という名前のシーケンシャルファイルにデータを追加して書き込むために，ファイル番号を２としてOPENする．

⑧　OPENされているファイルをすべて閉じる．

⑨　ファイル番号１のランダムファイルのN番目のレコードを読んで，バッファに入れる．

⑩　１番のドライブに入っているディスクの使用終了を宣言する．

第12章　プログラムの組み方

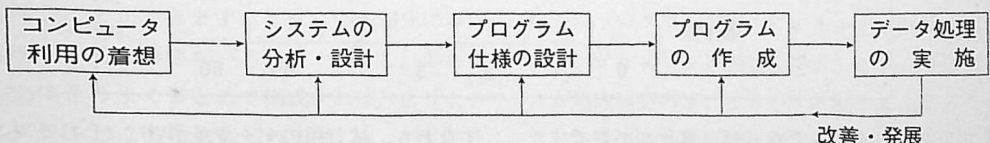
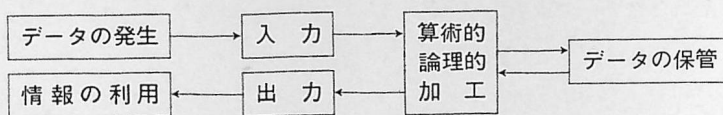
前章まではBASIC言語の文法を中心として解説を進めてきました．本章では，文法解説ではなく，プログラム作成に関するいくつかの事について述べるので，よいプログラムを作るための参考にしてください．

12・1　プログラム作成の手順

何かの仕事にコンピュータを利用しようとする場合，いきなりキーボードの前に座ってプログラムを作り始めようと思っても，そうはいきません．その前に，どんなプログラムを作るのかを決めなければなりません．さらにさかのぼれば，どのような仕事に，どんな目的でコンピュータを利用するのかを決めなければなりません．

一度限り使う簡単なプログラムならば別ですが，そうでなければ，上記のことをよく検討，立案することによって，より良いデータ処理システムを作り上げるようにすることが大切です．

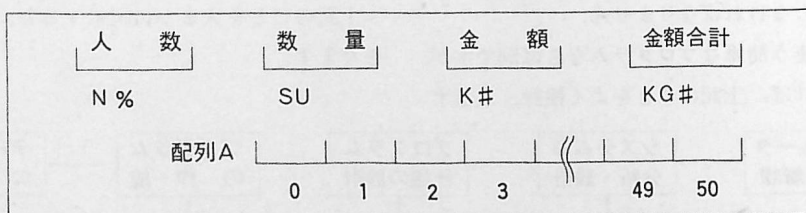
また，一旦作られたデータ処理システムを実際に動かし運用して行く段階においても，運用の結果を検討して，さらに使い易いシステムに改善して行くことも大切です．

上記のことを大まかに図示すると，次のようになります．

一般にデータ処理の過程は，次図に示すようにいくつかの機能部分から構成されているものと見ることができます．

システムの分析・設計の段階では，およそ次のような事項について調査・検討し，これから作ろうとするデータ処理システムの構想を立案します．

① コンピュータを適用する業務の内容．どんな情報が業務のために必要なのか，その情報を作り出すにはどんなデータが必要か．
② 処理されるべきデータの量や，加工処理の複雑さはどの程度か．
③ データを保管するためには，どのようなファイルが必要か．それはシーケンシャルファイルにするか，それともランダムアクセスファイルが必要か．
④ どのような機器構成が必要か．

システムとの構想がまとまったならば，次にプログラムの仕様を詳細に設計しなければなりません．設計すべき項目としては次のようなものがあります．

① 入出力すべきデータ項目の型や桁数あるいは編集形式
② ファイルに記録すべきデータ項目およびファイルの性格
③ 使用するコード体系
④ エラーチェックの基準
⑤ 計算処理の手順と内容を示す流れ図（フローチャート）
⑥ その他，プログラムが持つべき機能・性能の細目

プログラムの仕様細目がきまったならば，いよいよ実際にプログラムを組んでコンピュータに入れるわけです．具体的に変数名をきめ，今までに覚えたBASIC言語の知識を総動員して，プログラムを書いて行きます．変数の名前や型などを誤りなく使い分けるためには，例えば次図のような，記憶場所のレイアウトを紙に書いて，それを見ながらプログラムの命令を書いて行くことをおすすめします．

プログラムを組んだならば，試行が必要です．すなわち，試行用のデータを作り，それを入力し

て予期した結果が得られるかどうかをテストするわけです．テストデータは，可能性のあるいろいろな条件の場合をテストできるように，注意深く作ることが必要です．

試行中にプログラムのエラーが発生したならば，エラーの原因を調べてプログラムを修正します．これを「デバッグ」といいます．

プログラムのテストやデバッグが完了したならば，そのプログラムをカセットテープあるいはディスクにセイブして保存します．それと同時に「プログラム説明書」を作成することをおすすめします．

プログラム説明書には次のような事項を整理して書きます．

プログラム名（セイブしたときのファイル名），作成年月日，作成者名，
プログラムの機能，制限条件，実行時の操作方法，
その他，プログラム仕様を設計するときに決めた細目など．

プログラム説明書は，作成するのは面倒ですが，後日役に立つものです．

以上でプログラムを作成する手順は終了したわけです．あとは，それを実際に使ってデータ処理を行ない，必要に応じて改善し，さらに発展させて行くことになります．

12・2　プログラム作成上のくふう

良いプログラムを作りたいとは，誰でも望むことです．では良いプログラムとはどんなものかというと，これはそう簡単にはいいきれませんが，およそ次のようなことが目安になるでしょう．

確実に動く．（当然のことですが）
処理速度が速い．
記憶場所をむだ使いしない．
実行時に使い易い．
プログラムが短い．
プログラムの書き方がすなおでわかり易い．
プログラムの修正・拡張がし易い．

上記のそれぞれは，互いに重複する面もあるし，また矛盾する面もあります．そのどこに重点を置くかによって，プログラムの作り方が変わってくるし，でき上がったプログラムの構造にも差異が生じます．

前節で図示したように，プログラムは，入力・出力・加工・保管といった機能部分から構成されると見ることができます．これらの各部分における処理の方法・形態をいろいろくふうし，またそれら全体をうまくまとめて構成すれば，よりよいプログラムになると考えられます．

パーソナルコンピュータのひとつの特徴は，人間とコンピュータとの対話形式による進行です．この形式をうまく取り入れることによって，使い易いプログラムになります．

例えば，「メニュー」というやり方があります．レストランで客がメニューを見て注文するのと同様で，プログラムが持っているいくつかの処理機能を，ディスプレイ画面に表示します．つまりこれが処理機能のメニューです．操作者はそれを見て，自分の欲する処理を選んで，キーボードから

その指定を入力します．このようなメニュー方式の例は，次章のプログラム例の中で示します．

データをキーボードから入力する場合,INPUT文を用いると画面に疑問符（？）が現われます．しかし？だけでは，どんなデータを入れるべきなのか，入力形式（桁数など）はどうなのか，などが操作者にとってわかりにくいものです．そこで，適当なガイドメッセージを画面に表示することが有益です．

下記は，TIME＄に時刻をセットする場合の例です．（TIME＄については第6章（6･7）参照）

〔例1〕
```
100 INPUT "TIME (HH:MM:SS)="; T$
110 TIME$=T$
```

また入力時に，操作者の注意を促がすために，画面をブリンクさせたり，あるいはブザーを鳴らすといった方法も考えられます．下記はその例です．

〔例2〕
```
10 CONSOLE , , 0, 1 ……………………カラーモードにする
20 LOCATE 5, 2 ……………………画面位置を指定
30 LINE 2, 1 ……………………ブリンクを開始する
40 BEEP ……………………ブザーを鳴らす
50 INPUT "スウリョウ="; S……………………データを入力
60 LINE 2, 0 ……………………ブリンクを止める
```

この例では，画面の行番号2，列番号5の位置に「スウリョウ＝」というメッセージを表示して入力を要求します．このときブザーを1回ブーと鳴らし，さらに，キーボードからの入力操作が終るまで2番の行全体をブリンクさせます．

CONSOLE，LOCATE，LINEについては第8章ですでに説明しました．BEEPについてここで説明を加えておきます．次のものがBEEP文です．

BEEP <ブザースイッチ>

ブザースイッチは省略することができます．省略すると，ごく短い時間だけ，コンピュータに内蔵されているブザーが1回鳴ります．もし，

```
40 BEEP:BEEP:BEEP
```

とすれば，少し長く「ブーーー」と鳴ります．

ブザースイッチを1と書くと，ブザーが鳴り始めて，鳴り続けます．

ブザースイッチを0と書くと，ブザーが鳴り止みます．

〔例3〕
```
100 BEEP 1 ……………………鳴り始める  ┐
110 INPUT "スウリョウ="; S              ├ この間，鳴り続ける
120 BEEP 0 ……………………鳴り止む     ┘
```

次に，プログラムを見易くするために，REM文をあちらこちらに入れるという方法があります．これはプログラムの各部分に対する注釈として有益です．しかし，REM文の分だけプログラムは長くなり，したがってプログラムを記憶するための場所もそれだけ多くとられます．一長一短です．

プログラムを短くする方法のひとつは，コロン（：）でステートメントを区切ることです．例えば，

〔例４〕

```
10  A＝123
20  B＝456
30  P＝3.14159
```

前記のものを，コロンを使って１行にまとめれば，次のようになります．

```
10  A＝123：B＝456：P＝3.14159
```

また，実数型と比べて整数型のデータは，記憶場所も少なくてすみ，また演算速度は大きいという性質を持っています．したがって，整数型として処理できるものは，積極的に型を指定して用いた方が得策です．

以上いくつかのことについて断片的に述べてきましたが，これらはプログラム作成上のくふうのほんの一部に過ぎません．問題に応じて，よりよいプログラムを作るように考えてください．

第13章　応用プログラム作成の手引

本章では，応用プログラム（アプリケーションプログラム）を作成するときの参考として，いくつかの例を掲げて説明を加えます．

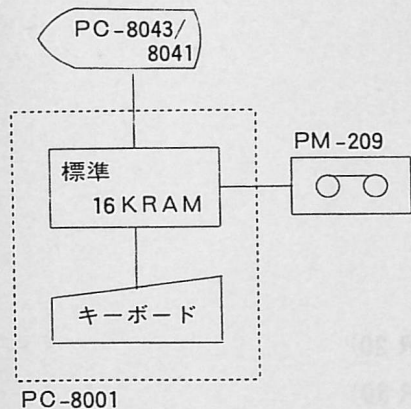
コンピュータの応用範囲は非常に広く，その処理内容は千差万別です．それぞれの仕事の内容に合わせて処理システムを設計し，よいプログラムを作り，パーソナルコンピュータを活用してください．

13·1　ABC分析プログラム

ここでは，簡単なABC分析のプログラムについて，解説します．御存知のように，ABC分析というのは，パレート図の応用で，ある複数の事象（たとえば商品とか不良の原因等）のなかのあるものを，重点管理するために，重点管理するものとそうでないものとに分けるための，ひとつの手法です．

さて，今ここでは，パン屋さんの，パンの種類別の売上のABC分析ということで，プログラム例を示します．全部で，15種類あるパンの売上金額を，キーボードより入力します．総売上高の75%を占める商品群をAランク，Aランクを除いて90%までを占める商品群をBランク，それ以下をCランクとします．

13·1·1　機器構成

このABC分析プログラムは，次のような機器構成で，実行が可能です．

- PC-8001 （標準16KRAM）
- PC-8043 （高解像度カラーディスプレイ）
  （PC-8041（グリーンディスプレイ））
- RM-209 （カセットテープ・レコーダ）

用意するディスプレイは，PC－8041であっても，A，B，Cのランク別に表示するキャラクタを変化させてやれば，見易くなります．RM－209は，この分析プログラムのセイブ，あるいは，ロードの為に，必要です．

13·1·2　処理の条件

a.　パンの種類（15種類）

食パン，厚切パン，ドックロール，バターロール，アンパン，コッペパン，ジャムパン，クリームパン，メロンパン，カステラサンド，コーヒ

ーロール，レーズンロール，アンドーナツ，クリームフライ，カップケーキ

b．　上記の順番で，売上金額の入力を求めます．１件入力する毎に，確認のステップを設けます．

c．　ファンクション・キーを次のように定義します．

f・1……／OK C_R，　f・2～f・4……NULL，　f・5……／NOC_R

d．　画面は，分析グラフ表示の時は，80×25，それ以外は，36×25で使います．

e．　売上金額入力の時，売上高／総売上高の割合表示の時および，分析グラフ表示の時の，ディスプレイのデザインは，次のようにします．

(売上金額入力)

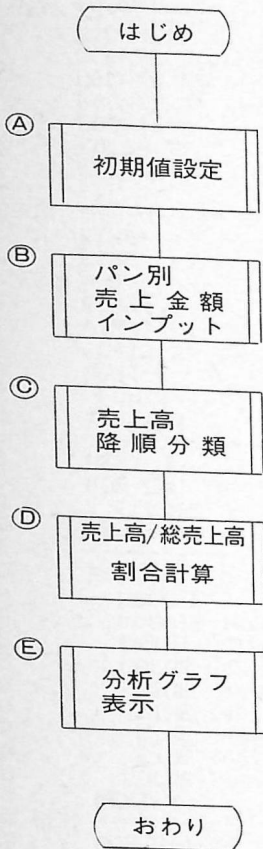
＊＊＊ ショウヒン ノ ABC ブンセキ ＊＊＊

	ショウヒン	キンガク	リツ	ルイセキ
1.	XXXXXXXXXX	99,999,999	99.9%	999.9%
2.				
3.				
15.	XXXXXXXXXX	99,999,999	99.9%	999.9%

（売上高/総売上高 割合の表示）

（分析グラフ表示）

13・1・3　処理の内容と流れ

このプログラムは，次のような処理で，構成されています．

Ⓐの初期値設定では，画面の定義，ファンクション・キーの定義，パンの商品名の設定などを行ないます．

Ⓑでは，Ⓐにおいて，あらかじめ設定された商品の順に，売上金額を，キーボードより，入力します．

Ⓒでは，15種のパンを，売上高の多い順に，並べかえます．

Ⓓでは，Ⓒで並べかえた順に，売上高／総売上高の割合を計算し，かつ，その割合を，累積していきます．

Ⓔでは，Ⓒで計算した割合を，横棒グラフで表示し，あわせて，A，B，Cのランクを判別出来るようにします．

13·1·4　プログラム

```
1000 REM *** ショウヒン ノ ABC ブンセキ ***
1010      WIDTH 36,25
1020      CONSOLE 0,25,0,1           ―画面の定義
1030      PRINT CHR$(12)
1040 REM ----------
1050      KEY1,"/OK"+CHR$(13)
1060      KEY2,""
1070      KEY3,""                    ―ファンクションキーの定義
1080      KEY4,""
1090      KEY5,"/NO"+CHR$(13)
1100 REM ----------
1110      DIM N$(15)  'ショウヒン メイ
1120      DIM K#(15)  'キンガク
1130      DIM R#(15)  'リツ
1140      DIM T#(15)  'ルイセキ
1150      DIM C(15)   'イロ
1160 REM ----------
1170      DATA "ショクパン     "
1180      DATA "アッギリ パン  "
1190      DATA "ドックロール   "
1200      DATA "バターロール   "
1210      DATA "アンパン       "
1220      DATA "コッペパン     "
1230      DATA "ジャム パン    "
1240      DATA "クリーム パン  "
1250      DATA "メロン パン    "
1260      DATA "カステラ サンド  "        ―パンの種類名のセント
1270      DATA "コーヒーロール   "
1280      DATA "レーズンロール  "
1290      DATA "アン ドーナツ   "
1300      DATA "クリーム フライ  "
1310      DATA "カップ ケーキ   "
1320 REM ----------
1330      FOR X=1 TO 15
1340        READ N$(X)
1350      NEXT X
1352 GOTO 2500
2000 REM ----------
2010      LOCATE 0,0:PRINT "*** ショウヒン ノ ABC ブンセキ ***";
2020      LOCATE 2,2:PRINT "キンガク イレテ クダサイ";
2030      FOR X=1 TO 15
2040        LOCATE 3,X+3:PRINT USING "##";X;:PRINT "."+N$(X);
2050        LOCATE 22,X+3:PRINT SPC(20);
2060        LOCATE 22,X+3:INPUT K#(X)
2070        LOCATE 22,X+3:PRINT USING "########,";K#(X);
2080          GOSUB 5000 ――――――――――キー入力の確認サブルーチンへ。
2090            IF KY$="/NO" THEN GOTO 2050
2100        S#=S#+K#(X)
2110      NEXT X
3000 REM ----------
3010      FOR I=1 TO 14
3020        FOR J=I+1 TO 15
3030          IF K#(J)<=K#(I) THEN GOTO 3050       ―売上金額降順(大きい順)に並びかえ
3040          SWAP K#(J),K#(I):SWAP N$(J),N$(I)      パン名も並びかえる。
3050        NEXT J
3060      NEXT I
3070 REM ----------
3080      PRINT CHR$(12)
3090      LOCATE 0,0:PRINT "*** ショウヒン ノ ABC ブンセキ ***";
3100      LOCATE 0,2:PRINT "    ショウヒン            キンガク  リツ   ルイセキ";
3120      FOR X=1 TO 15
3130        LOCATE 0,X+3:PRINT USING "##";X;:PRINT "."+N$(X);
3140        LOCATE 14,X+3:PRINT USING "########,";K#(X);
```

```
3150          R#(X)=K#(X)/S#*100 ─┐
3160          T#(X)=T#(X-1)+R#(X)─┴──────────売上高／総売上高の計算，率の累計
3170          LOCATE 24,X+3:PRINT USING "##.#%";R#(X);
3180          LOCATE 30,X+3:PRINT USING "###.#%";T#(X);
3190        NEXT X
3200        LOCATE 14,22:INPUT "ﾂｷﾞ ﾆ ｽｽﾐ ﾏｽｶ";KY$
3210        IF KY$<>"/OK" THEN GOTO 3200
4000 REM ----------
4010        WIDTH 80,25
4020        LOCATE 5,0:PRINT "*** A B C  ﾌﾞﾝｾｷ  ｸﾞﾗﾌ  ***";
4030        LOCATE 25,2:COLOR 2:PRINT "ｱｶ ... A";
4040        LOCATE 36,2:COLOR 6:PRINT "ｷｲﾛ ... B";
4050        LOCATE 49,2:COLOR 5:PRINT "ﾐｽﾞｲﾛ ... C";
4060        COLOR 7
4070        LOCATE 22,3:PRINT "0   10   20   30   40   50   60   70   80   90  100"
;
4090 REM ----------
4100        LINE (44,19)-(44,81),PSET  ←────グラフのＹ軸
4110        LINE (45,81)-(144,81),PSET ←────グラフのＸ軸
4120 REM ----------
4130        FOR I=1 TO 15                    ─┐
4140         IF T#(I)>75 THEN GOTO 4170       │
4150         C(I)=2   ←────Ａランク(赤色)     │
4160        NEXT I                            │
4170         FOR J=I TO 15                    │
4180          IF T#(J)>90 THEN GOTO 4210      ├色の設定
4190          C(J)=6  ←────Ｂランク(黄色)     │
4200        NEXT J                            │
4210        FOR L=J TO 15                     │
4220          C(L)=5  ←────Ｃランク(水色)     │
4230        NEXT L                           ─┘
4240 REM ----------
4250        FOR X=1 TO 15
4260          LOCATE 0,X+4:PRINT USING "##";X;:PRINT "."+N$(X);
4270          LOCATE 16,X+4:PRINT "(";
4280          PRINT USING "##.#";R#(X);:PRINT ")";
4285          COLOR C(X)
4290           LINE (46,4*(X-1)+20)-(INT(R#(X))+46,4*(X-1)+20),PSET ─┐
4300           LINE (46,4*(X-1)+21)-(INT(R#(X))+46,4*(X-1)+21),PSET  │１キャラクタ分の
4310           LINE (46,4*(X-1)+22)-(INT(R#(X))+46,4*(X-1)+22),PSET  │横棒グラフ
4320           LINE (46,4*(X-1)+23)-(INT(R#(X))+46,4*(X-1)+23),PSET ─┘
4330          COLOR 7
4335        NEXT X
4340 REM ----------
4350        FOR X=1 TO 10000:NEXT X ←────WAIT
4360        END
5000 REM ----------
5010        LOCATE 14,22:PRINT "ｶｸﾆﾝ ｼﾃ ｸﾀﾞｻｲ";      キー入力確認サブルーチン
5020        LOCATE 15,23:PRINT SPC(10);
5030        LOCATE 15,23:INPUT "/OK?  /NO? ";KY$
5040          IF KY$<>"/OK" AND KY$<>"/NO" THEN GOTO 5020
5050            LOCATE 14,22:PRINT SPC(20);
5060            LOCATE 15,23:PRINT SPC(20);
5070        RETURN
```

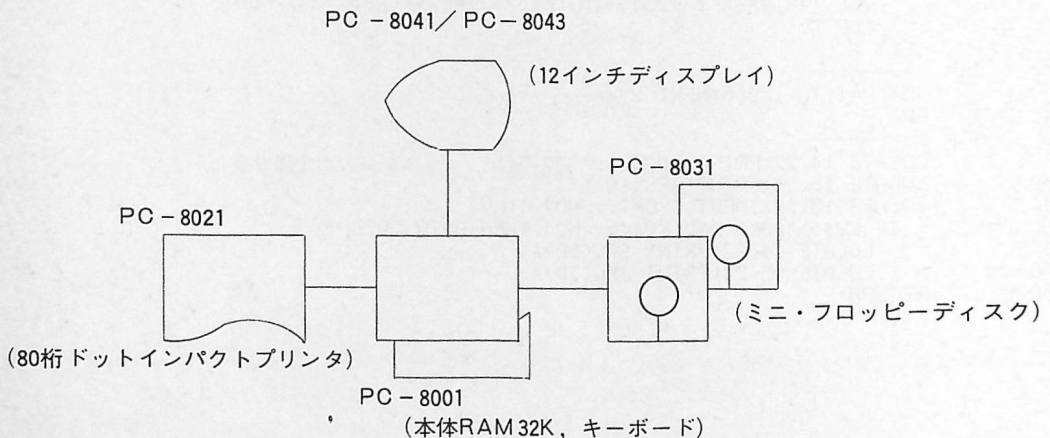
13·2　給与計算システム

パーソナルコンピュータを使った事務処理への応用例といえば，すぐあげられるものの中に，給与計算業務があります．ここでは，この給与計算のプログラムについて，解説します．ここにあげる例は，実際に，次のような概要の企業で，使われているものです．この会社では，社長一人で，延2日間の時間をかけて，計算していましたが，パーソナルコンピュータを使うことによって，社長以外の人間に給与計算業務を移行させる，計算を半日で行なうことが出来るようになりました．

（業種）　めん類製造販売
　　　　　食品材料販売
（資本金）　3600万円
（従業員数）　120名
（年商）　9億円

この給与計算の基本的な考え方は，基本給とか△△手当とかの固定的なデータを入れておくファイル（給与マスタ）を作っておき，残業とか，欠勤とかの毎月変化のあるデータは，毎月キー入力してやり，給与計算を行なう，というものです．

13·2·1　機器構成

このシステムの機器構成は，下の通りです．PC－8001の主記憶容量は，32Kバイトです．フロッピーディスクは，2ドライブ使用します．ディスプレイは，カラーでも，グリーンでも，どちらでもかまいません．（プログラムは，カラーモードで，書かれています．）プリンタは，1行80桁のもので，給与明細書（13·2·7参照）等は，複写が必要なため，ドットインパクトプリンタが使われています．

13・2・2　基本機能

このシステムの基本的な機能は，次の通りです．

a)　対象人員　　最大　199人
b)　給与マスタ管理
c)　給与明細ファイル作成
d)　給与明細書作表
e)　金種表／銀行別振込合計表　作表
f)　部門別給与項目集計表　作表
g)　給与明細ファイルの訂正
h)　給与明細内容の給与マスタへの書き込み

この本では，d)以下は考え方を述べるだけにとどめ，詳しい説明は省略します．c)で給与明細ファイルさえ出来れば，d)，e)，f)は，作表だけの問題となります．各処理の概要については，次節で説明します．また給与マスタとは，従業員毎に，基本給，職務給等の固定的データを貯えておくものです．マスタというのは，ある程度の期間，変動のないデータを貯えている，その業務の基本となるデータファイルのことです．たとえば，販売商品の単価を貯えているデータファイルは，商品単価マスタと呼んだりします．給与明細ファイルは，給与マスタの内容と，キー入力される月々の変動データとから，計算されて作成されるものですから，単に，ファイルと呼びます．

13・2・3　プログラム構成

プログラム構成は，図のようになっています．″で囲まれているのは，プログラム番号（ファイル名の一種）です．このように，R10からR80までのプログラムで，このシステムは構成されています．それぞれのプログラム間には，ある一定の前後関係がありますが，それは，次節のシステムフローを参考にして下さい．たとえば，わかりきったことですが，R30が実行されていない（給与明細ファイルが出来ていない）のに，R40からR80までのプログラムを実行することは出来ません．R10とR20は，いつでも，必要な都度，実行することが出来ます．また，R30実行前に，給与マスタを最新の状態にするために，応々にして，R10を実行する場合が出てきます．

“MENU” という名のプログラムを実行させることによって，各プログラムを選択して，実行させます．このプログラムは，次図のようなメニュ画面を表示し，番号を入力することにより，各プログラムを選択するものです．この方式は，いろいろな場合に応用がききますので，コーディング例を示しておきます．このプログラムが，その日（電源投入後）初めて起動された時には，年月日と時刻のセットが出来るようにしておきます．

また，このプログラムにおいて，この給与計算システムで統一的に使うファンクションキーを次のように定義します．

f・1 ……　／OK RETURN
f・2 ……　／ADD RETURN
f・3 ……　／NEXT RETURN
f・4 ……　／END RETURN
f・5 ……　／NO RETURN

次に，プログラム番号（ファイル名）の付け方ですが，次のような付け方はどうでしょうか．

たとえば，この会社では，頭1桁目を英字にして，2桁目は適当に番号をつけて，3桁目を改訂番号にしています．

処理選択プログラム "MENU"

(プログラム番号)	(プログラム名)	(プログラム概要)
"R 10"	給与マスタ管理	給与マスタレコードの照会，削除，追加，変更を行ないます．
"R 20"	給与マスター覧表作成	給与マスタレコードの個人別リストを作成します．
"R 30"	給与計算（給与明細ファイル作成）	毎月変動するデータ（勤怠データや控除データ）をキー入力することにより，マスタの内容を参照しつつ計算を行ない，明細ファイルを作成します．
"R 40"	給与明細書　作表	指定帳票に，給与明細をプリントアウトします．
"R 50"	金種表／銀行別振込合計表　作表	金種表，あるいは，銀行別の給与振込合計表を作成します．
"R 60"	部門別給与項目集計表　作表	部門別・給与項目別に，その金額の集計表を作成します．
"R 70"	給与明細ファイル訂正	給与計算後，明細書発行後あるいは給与支払後，誤り発見の場合，給与明細ファイルの訂正を行ないます．
"R 80"	給与明細書き込み	給与明細ファイルの取得休暇数，当月勤怠データ，当月支給総額を給与マスタへ書き込んで，翌月の給与計算に備えます．

MENU)

1．ネンガッピ　　99/99/99

2．ジコク　　　99:99:99

/OK?　/NO?　ラインNO．?

「MENUプログラムコーティング例」

```
10000 REM --- MENU PROGRAM ---
10010      WIDTH 36,25
10020      CONSOLE 0,25,0,1
10030      KEY1,"/OK"+CHR$(13)
10040      KEY2,"/ADD"+CHR$(13)
10050      KEY3,"/NEXT"+CHR$(13)
10060      KEY4,"/END"+CHR$(13)
10070      KEY5,"/NO"+CHR$(13)
10080 REM --- DATE$,TIME$ SET ---
10090      IF DATE$<>"79/01/01" THEN GOTO 10250
10100      PRINT CHR$(12)
10110      LOCATE 3,2
10120      PRINT "(MENU)";
10130      GOSUB 10570
10140      GOSUB 10720
10150      LOCATE 16,22
10160      PRINT "/OK? /NO? ﾗｲﾝNO.?";
10170      LOCATE 26,23:PRINT SPC(5);
10180      LOCATE 26,23:INPUT KY$
10190      IF KY$="/NO" THEN GOTO 10100
10200      IF KY$="/OK" THEN GOTO 10250
10210      IF KY$="1" THEN GOSUB 10570:GOTO 10150
10220      IF KY$="2" THEN GOSUB 10720:GOTO 10150
10230      GOTO 10150
10240 REM --- MENU DISPLAY ---
10250      PRINT CHR$(12)
10260      LOCATE 18,0:PRINT DATE$;
10270      LOCATE 27,0:PRINT TIME$;
10280      LOCATE 3,2:PRINT "(MENU)";
```

10030〜10070：ファンクションキーの定義

10090：DATE$とTIME$のセットルーチンは，初回だけしか流れないようにするための判定.

10130：DATE$セットのサブルーチンへ.

10140：TIME$セットのサブルーチンへ.

10210〜10220：1あるいは2が入力されれば，それぞれ年月日，時刻の訂正が出来るようにします.

```
10290      LOCATE 5,4:PRINT "ｷｭｳﾖ ﾏｽﾀ ｶﾝﾘ";
10300      LOCATE 5,8:PRINT "ｷｭｳﾖ ﾒｲｻｲ ﾌｧｲﾙ ｶﾝｹｲ";
10310      LOCATE 5,13:PRINT "ｻｸﾋｮｳ ｶﾝｹｲ";
10320      LOCATE 6,5:PRINT "1. ﾄｲｱﾜｾ(R10)"
10330      LOCATE 6,6:PRINT "2. ｻｸﾋｮｳ(R20)";
10340      LOCATE 6,9:PRINT "3. ｷｭｳﾖ ｹｲｻﾝ  (R30)";
10350      LOCATE 6,10:PRINT "4. ﾃｲｾｲ        (R70)";
10360      LOCATE 6,11:PRINT "5. ﾏｽﾀ ﾍ ｶｷｺﾐ (R80)";
10370      LOCATE 6,14:PRINT "6. ｷｭｳﾖ ﾒｲｻｲｼｮ        (R40)";
10380      LOCATE 6,15:PRINT "7. ｷﾝｼｭﾋｮｳ            (R50)";
10390      LOCATE 6,16:PRINT "8. ﾌﾞﾓﾝﾍﾞﾂ ｼｭｳｹｲﾋｮｳ (R60)";
10400      LOCATE 14,22:PRINT "ﾌﾟﾛｸﾞﾗﾑ ｾﾝﾀｸ ｼﾃ ｸﾀﾞｻｲ";
10410      LOCATE 26,23:PRINT SPC(5);
10420      LOCATE 26,23:INPUT KY$
10430        IF KY$="/END" THEN END
10440        IF LEFT$(KY$,1)<"1" THEN GOTO 10410
10450        IF LEFT$(KY$,1)>"8" THEN GOTO 10410
10460      KY=VAL(LEFT$(KY$,1))
10470 REM --- PROGRAM SELECT ---
10480      ON KY GOTO 10490,10500,10510,10520,10530,10540,10550,10560
10490      RUN "R10"
10500      RUN "R20"
10510      RUN "R30"
10520      RUN "R70"
10530      RUN "R80"
10540      RUN "R40"
10550      RUN "R50"
10560      RUN "R60"
10570 REM --- DATE$ INPUT ---
10580      LOCATE 6,4:PRINT "1.ﾈﾝｶﾞｯﾋﾟ";
10590      LINE 4,1
10600      LOCATE 17,4:PRINT SPC(20);
10610      LOCATE 17,4:INPUT YM$
10620        IF MID$(YM$,1,2)<"80" THEN GOTO 10580
10630        IF MID$(YM$,1,2)>"99" THEN GOTO 10580
10640        IF MID$(YM$,3,2)<"01" THEN GOTO 10580
10650        IF MID$(YM$,3,2)>"12" THEN GOTO 10580
10660        IF MID$(YM$,5,2)<"01" THEN GOTO 10580
10670        IF MID$(YM$,5,2)>"31" THEN GOTO 10580
10680      LINE 4,0
10690      DATE$=MID$(YM$,1,2)+"/"+MID$(YM$,3,2)+"/"+MID$(YM$,5,2)
10700      LOCATE 17,4:PRINT DATE$;
10710      RETURN
10720 REM --- TIME$ INPUT ---
10730      LOCATE 6,7:PRINT "2.ｼﾞｺｸ";
10740      LINE 7,1
10750      LOCATE 17,7:PRINT SPC(10);
10760      LOCATE 17,7:INPUT TM$
10770        IF MID$(TM$,1,2)<"00" THEN GOTO 10730
10780        IF MID$(TM$,1,2)>"24" THEN GOTO 10730
10790        IF MID$(TM$,3,2)<"00" THEN GOTO 10730
10800        IF MID$(TM$,3,2)>"60" THEN GOTO 10730
10810        IF MID$(TM$,5,2)<"00" THEN GOTO 10730
10820        IF MID$(TM$,5,2)>"60" THEN GOTO 10730
10830      LINE 7,0
10840      TIME$=MID$(TM$,1,2)+":"+MID$(TM$,3,2)+":"+MID$(TM$,5,2)
10850      LOCATE 17,7:PRINT TIME$
10860      RETURN
```

(10290〜10390) メニュウ画面の表示

(10420) 文字変数で受けているのは，“/END”が入力される場合もあるから，また，数値を入力するつもりが，誤って数値以外を入力しても，そこで，プログラムが止まらないようにするためです．

(10620〜10670) 入力された年月日の内容をチェックしています．

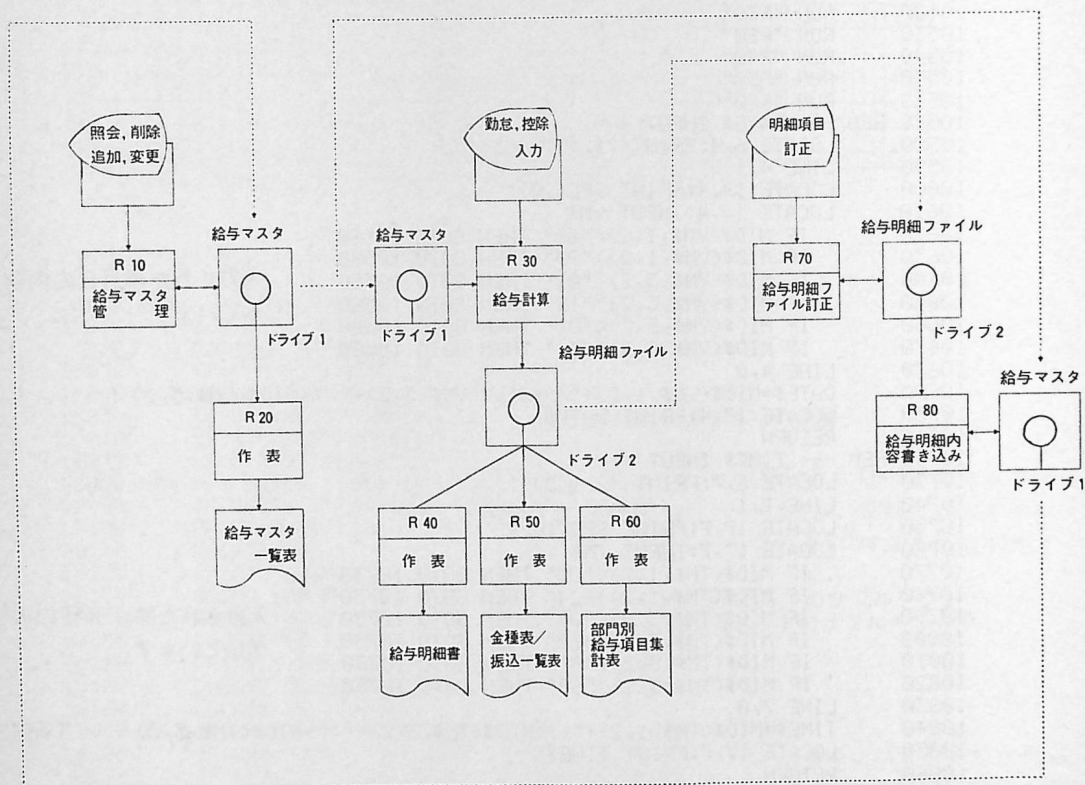
(10770〜10820) 入力された時，分秒の内容をチェックしています．

(10840) TIME$にセット．

13·2·4　システムの流れ

前節で説明したプログラムの構成を，処理の流れがわかるように図示したものが，システムフローと呼ばれるものです．ここでは，給与マスタ，給与明細ファイルが，どのドライブにセットされるべきかも，表示されています．給与マスタと給与明細ファイルがともに，複数の場所に図示されていますが，それらは，すべて同じもので，処理の流れをわかりやすくするために，そうなっているものです．

このシステムは，プログラムと給与マスタを1枚のフロッピーディスクに同居させドライブ1にセットする．また給与明細ファイルは別のフロッピーディスクに入れてドライブ2にセットする，という考えになっています．

処理の流れ（システムフロー）

13・2・5　ファイルデザイン

このシステムでは，今まで述べてきたように，　2つのファイルを使っています．

給与マスタ	"MSTR01"	1人分／1レコード／1セクタ
給与明細ファイル	"MSAI01"	1人分／1レコード／1セクタ
	(ファイル名)	

この2つのファイルの項目のデザインは，次のようになっています．項目の後につけられている略号は，その項目の桁数，および文字型，数値型の別をあらわしているものです．

C－XX；文字扱い，桁数
I；数値扱い2バイト文字列
S；数値扱い4バイト文字列
D；数値扱い8バイト文字列

〔給与マスタ，"MSTR０１"〕

所属コード	従業員コード	氏　　名	年	月	日				
			(1) 生年月日	(2) 入社年月日	(3) 退社年月日	(4) (予備)	(5) (予備)	(1) 役職	(2) 振込銀行
C－6	C－6	C－15	S	S	S	S	S	I	I

			区	分				日数／回数					前月支給総額
(3) 配偶者	(4) 扶養親族数	(5) 障害等の老数	(1) 税表区分	(2) (予備)	(3) (予備)	(4) (予備)	(5) (予備)	(1) 前年度休暇残	(2) 当年度休暇残	(3) 欠勤日数	(4) 遅刻回数	(5) 早退回数	
I	I	I	C－1	C－1	C－1	C－1	C－1	I	I	I	I	I	D

(1) 基本給	(2) 職能手当	(3) 役付手当	(4) 家族手当	(5) 住宅手当
S	S	S	S	S

手				当								
(6) 通勤手当	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(1) 健康保険料	(2) 厚生年金保険料	(3) 住民
S	S	S	S	S	S	S	S	S	S	S	S	S

	控						除					
税(1)	(4) 住民税(2)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)
	S	S	S	S	S	S	S	S	S	S	S	S

(予備)
C－56

〔所属コード〕　文字６桁

所属部門識別のためにあります．この会社の場合は，次のようにつけられています．

□ □ □ □ □ □

部　課　係　未使用

〔従業員コード〕　文字６桁

従業員一人一人につけられたコードです．給与マスタ管理，および給与計算等は，すべて，このコードをキーにして行なわれます．この会社では，次のようなつけ方をしています．

〔退社年月日〕　単精度

従業員が退社しても，その年度内は，給与マスタ上より，そのレコードを削除しないでおいて，退社年月日の入っている，入っていないで，退職者かどうか，判断するものです．年度がわりの時に，退社年月日の入っているレコードだけ，まとめて削除します．

〔役職コード〕　整数

この会社の場合，次のように設定されています．

1……役員
2……役員兼部長
3……部長
4……次長
5……課長
6……係長
7……主任
8……一般
9……パート

〔前月支給総額〕　倍精度

当月の支給総額を計算した時，前月のそれと比較して，著しい差がないかどうか，チェックのための項目です．

〔手当〕　単精度

(1)〜(6)までは，項目名が決められていますが，(7)〜(15)までは，項目名を自由に設定出来るようにしてあります．

〔控除〕　単精度

(1)〜(4)までは，項目名が決められていますが，(5)〜(15)までは，項目名を自由に設定出来るようにしてあります．

〔給与明細ファイル, "MSAI 0 1"〕

所属コード	従業員コード	氏　　名	日数					時間			
			(1) 出日勤数	(2) 欠日勤数	(3) 休日暇数	(4) 残休日暇数	(5) (予備)	(1) 基準内労働時間	(2) 残業時間	(3) 深夜公出時間	(4) 遅
C－6	C－6	C－15	I	I	I	I	I	S	S	S	

		回数					基本給	手						
刻時間	(5) 早退時間	(1) 予備	(2) 予備	(3) 予備	(4) 遅回刻数	(5) 早回退数		(1) 職能手当	(2) 時間外手当	(3) 精勤手当	(4) 役付手当	(5) 家族手当	(6) 住宅手当	(7) 通
S	S	I	I	I	I	I	D	S	S	S	S	S	S	

当													
勤手当(課税)	(8) (予備)	(9) (予備)	(10) (予備)	(11) (予備)	(12) (予備)	(13) (予備)	(14) (予備)(非課税)	(15) 通勤手当(非課税)	(1) 健康保険料	(2) 厚生年金保険料	(3) 雇用保険料	(4) 欠勤控除
S	S	S	S	S	S	S	S	S	S	S	S	S

控除												
	(5) (予備)	(6) 所得税	(7) 市民税	(8) (予備)	(9) (予備)	(10) (予備)	(11) (予備)	(12) (予備)	(13) (予備)	(14) (使用しない)	(15) (使用しない)	(1) 社会保険額合
	S	S	S	S	S	S	S	S	S	S	S	D

合計						
計	(2) 課税対象額	(3) ㉝以降差引額合計	(4) 支給総額	(5) 差引総額	(6) 当月支払総額	(予備)
	D	D	D	D	D	C－9

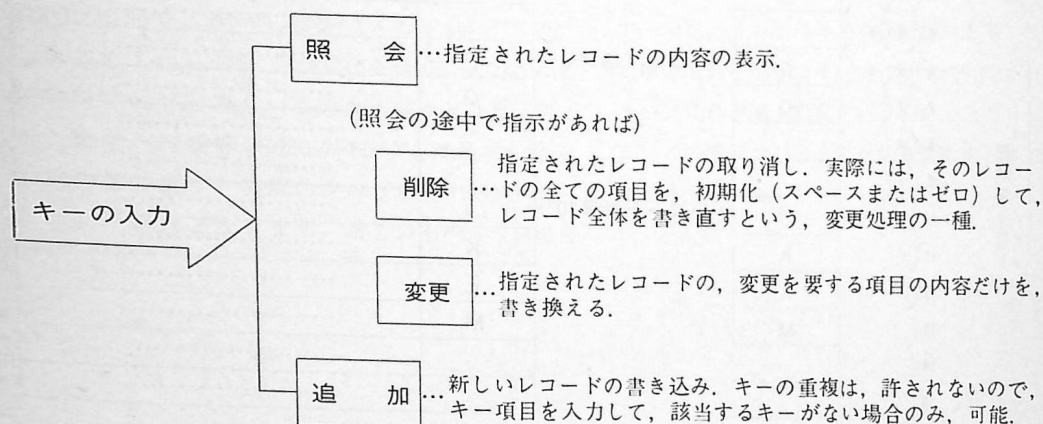
年　月
C－4

給与明細書（13・2・8参照）

13・2・6　給与マスタ管理プログラム(R10)

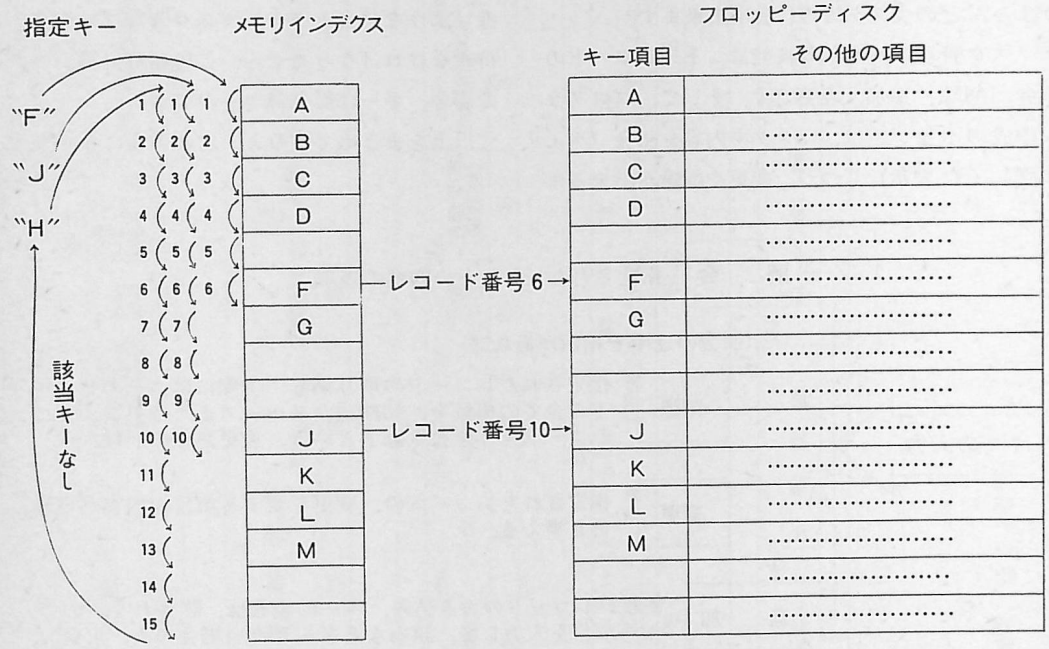
前節までにおいて，この給与計算システムの概要を説明してきましたが，この節は，具体的なプログラムについて，解説します．この給与マスタ管理プログラムは，従業員毎に，その給与マスタレコードの照会，削除，追加，変更を行なうものです．このマスタ管理プログラムの考え方は，他のほとんどのマスタ管理に応用出来ます．

マスタ管理の一般的な機能は，そのレコードの照会，削除，追加，変更です．そして，プログラム作成のポイントは，マスタの内容を照会（ディスプレイに表示）しつつ，変更や削除が出来るようにするところにあります．マスタを管理（照会，削除，追加，変更）するためには，キーとなる項目が必要です．キーというのは，欲しいレコードを選択するための，特定の項目のことです．ですから，マスタの中に，同じキーを持つレコードが，複数あることは許されません．キーの重複を許さないようなチェックを，マスタ管理プログラムは，持たなければなりません．この給与計算システムの場合，キーは従業員コードです．

以上をまとめてみると，次のようになります．

次に，メモリインデクスという考え方について，説明します．マスタ管理の場合，キーを指定しますが，実際に，フロッピーディスク上のレコードを読み書きする場合，レコード番号がわからなくてはなりません．指定されたキーを持つレコードは，何番目のレコードなのかを，知る必要があるわけです．そのために，マスタ管理の諸々の処理に先立ち，そのマスタの全レコードを読んで，キー項目の部分だけを，メモリ上に持って来ます．今の場合，従業員コードだけの集合を，メモリ上につくることになります．これを，メモリ上の索引ということで，一般的に，メモリインデクスと呼んでいます．

キーが指定されると，まず，メモリインデクスの中で，同じものがあるかどうかをみて（もしあれば，それは，何番目にあらわれたか，これがレコード番号になります)，次に，実際に，フロッピーディスクへの処理に移ります．

たとえば，指定されたキーが“F”の場合（次図参照）メモリインデクスを，はじめから順に読んで行きます．キー“F”とメモリインデクスの内容が，6番目に一致しましたので，レコード番

号6ということで，フロッピーディスク処理に入ります．“F”というキーをもつレコードの内容を表示するなり（照会），項目の書き換え（変更，削除）を行なうことが出来ます．いま，新規にレコードを追加しようとして，そのキーが“J”だった場合，“J”はすでに登録されていますので，キーの重複ということになり，追加処理をしてはいけません．キーをかえるか，すでに登録されている“J”の内容の変更ということで対応しなければなりません．また，“H”というキーが指定された場合，メモリインデクスを最後まで読んでいっても，そのキーはありませんので，該当キーなし（アンマッチ）ということになります．この場合，照会，削除，変更は，当然出来ませんが，追加処理は出来ます．今の場合，5番目のレコードが空いていますので，レコード番号5で，フロッピーディスクに書き込む（追加）ようにします．

次にメモリインデクスの使い方のプログラムコーディング例を示します．

〔メモリインデクスの作成〕

```
10010 DIM  CB$(200)
10020 MOUNT  1
10030 OPEN  "MSTR01"  AS  #1
10040 FIELD  #1, 6  AS  MA$, 6  AS  MB$
10050 FOR  X=1  TO  200
10060 GET  #1, X
10070 CB$ (X)=MB$
10080 NEXT  X
  ⋮
```

キーである従業員コードは，行番号 10040 のFIELD文で，ＭＢ＄という名で６桁と定義されています．行番号 10050 から 10080 の間で，レコード番号を１から 200 まで変化させて，給与マスタレコードを読んできて，従業員コードを配列 CB$ にセットしています．CB$ は，行番号 10010 で，200 の大きさであると定義されており，これが，メモリインデクスになります．

〔メモリインデクスの検索〕

```
25000 GOSUB 30540
25010 FOR X=1 TO 200
25020 IF CB$(X)=K3$ THEN GO TO △△△△△
25030 NEXT X
25040 ..................
   ⋮
```

行番号 30540 から始まるサブルーチンで，キーの指定（従業員コードの入力）が行なわれています．Ｋ３＄がその変数名です．行番号 25010 から 25030 の間で，配列ＣＢ＄（メモリインデクス）の添字を１から 200 まで変化させて，入力された従業員コードと等しいものがあるかどうか，調べています．行番号 25020 のＩＦ文を満足すれば，その時のＸの値が，実際のフロッピーディスク上のレコード番号となります．等しいものがない場合は，該当キーなし（アンマッチ）ということで，行番号 25040 に降りてきます．

以上が，メモリインデクスの考え方，使い方です．次に，この給与マスタ管理プログラム全体の処理の流れを示します．次に示す流れ図を見て下さい．これが唯一の型というわけではありませんが，この流れで，ほとんどの業務のマスタ管理は出来ると思います．

理の流れ)

13·2·7　給与マスター覧表作成プログラム（R20）

このプログラムは，従業員毎に，その給与マスタレコードの内容を，ある一定のフォーマットに従って，作表するものです．従業員コードが指定されればそのコード以降から，指定がなければ第1件目から，作表を開始します．作表の終了は，ファイルの終り，あるいはSTOPコマンドによって行ないます．

1ページに，1人分を次のようなデザインで作表して行きます．

2 << ｷｭｳﾖ ﾀﾞｲﾁｮｳ ｲﾁﾗﾝ ﾋｮｳ >> (99/99/99) PAGE ###

4 ｼｮｿﾞｸ ｼﾞｭｳｷﾞｮｳｲﾝ ｺｰﾄﾞ ｼﾒｲ

DATESです。

6 XXXXXX XXXXXX XXXXXXXXXXXXXXX

ページカウンタを設一人分作表する毎に行きます。

10 <ﾈﾝｶﾞｯﾋﾟ>

年，月，日の間に／を入れて区切るようプログラムで編集します。

11 4. ｾｲﾈﾝ 99/99/99
12 5. ﾆｭｳｼｬ 99/99/99
13 6. ﾀｲｼｬ 99/99/99
14 7. ﾈﾝｶﾞｯﾋﾟ(4) 99/99/99
15 8. ﾈﾝｶﾞｯﾋﾟ(5) 99/99/99

17 <ｺｰﾄﾞ> <ｸﾌﾞﾝ>
18 9. ﾔｸｼｮｸ ## 14. ｾﾞｲﾋｮｳ X
19 10. ﾌﾘｺﾐ ｼﾞﾝｺｳ ## 15. ｸﾌﾞﾝ(2) X
20 11. ﾊｲｸﾞｳｼｬ ## 16. ｸﾌﾞﾝ(3) X
21 12. ﾌﾖｳｶｿﾞｸｽｳ ## 17. ｸﾌﾞﾝ(4) X
22 13. ｺｰﾄﾞ(5) ## 18. ｸﾌﾞﾝ(5) X

24 <ﾙｲｾｷ ｺｳﾓｸ>
25 19. ｷｭｳｶｻﾞﾝ(ｾﾞﾝ) ## 24. ｼｷｭｳｿｳｶﾞｸ(ｾﾞﾝ) #########
26 20. ｷｭｳｶｻﾞﾝ(ﾄｳ) ##
27 21. ｹｯｷﾝ ﾆｯｽｳ ##
28 22. ﾁｺｸ ｶｲｽｳ ##
29 23. ｿｳﾀｲ ｶｲｽｳ ##

30 LPRINT USING命令を使って，右詰になるように また，3桁ごとに，が入るように編集します。

31 <ﾃｱﾃ>
32 25. ｷﾎﾝｷｭｳ #########
33 26. ｼｮｸﾉｳ #########
34 27. ﾔｸﾂｷ #########
35 28. ｶｿﾞｸ #########
36 29. ｼﾞｭｳﾀｸ #########

38 30. ﾂｳｷﾝ #########
39 31. ﾃｱﾃ(7) ######### 35. ﾃｱﾃ(11) #########
40 32. ﾃｱﾃ(8) ######### 36. ﾃｱﾃ(12) #########
41 33. ﾃｱﾃ(9) ######### 37. ﾃｱﾃ(13) #########
42 34. ﾃｱﾃ(10) ######### 38. ﾃｱﾃ(14) #########
39. ﾃｱﾃ(15) #########

44 <ｺｳｼﾞｮ>
45 40. ｹﾝｺｳ ﾎｹﾝ #########
46 41. ｺｳｾｲ ﾈﾝｷﾝ #########
47 42. ｼﾞｭｳﾐﾝｾﾞｲ(1) #########
48 43. ｼﾞｭｳﾐﾝｾﾞｲ(2) #########
49 44. ｺｳｼﾞｮ(5) #########

51 45. ｺｳｼﾞｮ(6) ######### 50. ｺｳｼﾞｮ(11) #########
52 46. ｺｳｼﾞｮ(7) ######### 51. ｺｳｼﾞｮ(12) #########
53 47. ｺｳｼﾞｮ(8) ######### 52. ｺｳｼﾞｮ(13) #########
54 48. ｺｳｼﾞｮ(9) ######### 53. ｺｳｼﾞｮ(14) #########
55 49. ｺｳｼﾞｮ(10) ######### 54. ｺｳｼﾞｮ(15) #########

58 ここまでプリントしたら，改頁します。

この作表プログラムで注意すべきことは，1人分のマスタレコードを66行単位で作表するというところにあります．55行目でリストは終っていますので，残り11行分を改行して，次のレコードは，新たなページの2行目から，作表するようにしなければなりません．そのためには，最後の行（55行目）をプリントした後に，

```
LPRINT CHR$(&H8C)
```

を実行してやれば，次のページの頭まで，改行されます．

また，25行目のプリントは，次のように，ひとつのプリント命令で，記述することが出来ます．前年度休暇残，および前月支給総額の変数を，それぞれQ%，Q#とします．

```
LPRINT USING "␣␣␣␣␣␣␣␣␣␣␣ 19. キュウカザン（ゼン）␣␣␣##␣␣␣␣␣␣
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ 24. シキュウソウガク（ゼン）#########,"; Q
%; Q#
```

13·2·8　給与明細ファイル作成プログラム（R 30）

このプログラムは，毎月の変動データ（欠勤，遅刻，臨時控除等）を，キー入力することにより，給与マスタレコードを参照しつつ，給与の計算を行なうものです．次のような給与支払明細書を，給与明細書作表プログラム（R 40）で作るので，給与明細ファイルのデザインは，これを意識したものになっています．（13·2·5給与明細ファイルデザイン参照．）

給与支払明細書

年　月	所属コード	従業員コード	氏　名

①出勤日数	②欠勤日数	③休暇日数	④基準内労働時間	⑤早出残業時間	⑥深夜公出時間	⑦遅刻時間	⑧早退/就業時間	⑨	⑩	⑪	⑫休暇残

支給額内訳	⑬基本給	⑭職能手当	⑮時間外手当	⑯精勤手当	⑰役付手当	⑱家族手当	⑲住宅手当	⑳通勤手当(課税)
	㉑　手当	㉒　手当	㉓　手当	㉔　手当	㉕　手当	㉖　手当	㉗(非課税)　手当	㉘通勤手当(非課税)
差引額内訳	㉙健康保険料	㉚厚生年金保険料	㉛雇用保険料	㉜社会保険額合計	㉝欠勤控除	㉞　差引	㉟課税対象額	㊱所得税
	㊲住民税	㊳　差引	㊴　差引	㊵　差引	㊶　差引	㊷　差引	㊸　差引	㊹㊲以降差引額計

㊺支給総額
㊻差引総額
㊼支払総額

（ノーカーボン2枚複写1部…個人，1部…会社控）

給与明細ファイルの各項目には，キー入力により決まるもの，給与マスタから持ってくるもの，計算によるものがあります．手当，控除の予備欄は，どれによるかは，各会社により異なります．

〔キー入力による項目〕

従業員コード，出勤日数，欠勤日数，休暇日数，残業時間，深夜・公出時間，遅刻時間，早退時間，遅刻回数，早退回数，手当（14，非課税），控除（5），基準内労働時間

〔給与マスタからの項目〕

所属コード，氏名，基本給，職能手当，役付手当，家族手当，住宅手当，通勤手当，健康保険料，厚生年金保険料，住民税

〔計算または判定による項目〕

休暇残日数，時間外手当，精勤手当，通勤手当，雇用保険料，欠勤控除，所得税，社会保険額合計，課税対象額，㉝以降差引額合計，支給総額，差引総額，当月支払総額，年月

以上で，この節を終りますが，100人前後の給与計算を行なう場合，留意すべき点は，給与マスタ管理プログラムの操作の容易性にあると思います．

13·3　在庫管理システム

このシステムは，主として，卸売業での在庫管理（数量管理）を行なうことを，目的としています．このシステムを動作させるための機器構成は，前節の給与計算システムの場合と同様です．なお，この在庫管理システム例は，日本電気（株）製の在庫管理システム設計書より転載させて頂いたものです．その要点を紹介します．

13·3·1　基本機能

a）管理品目数　128件
b）入庫品目取引件数　256件
c）出庫品目取引件数　256件
d）入／出庫処理
e）発注処理
f）在庫一覧表　作成
g）発注点切れ表　作成
h）品目別入出庫明細表　作成
i）入庫先別入庫明細表　作成
j）出庫先別出庫明細表　作成

13·3·2　プログラム構成

プログラム構成は，次のようになっています．

それぞれのプログラムの処理概要は，次の通りです．

a) 日時設定（"DTSET.BS"）

処理開始にあたって，処理日および開始時刻を初期セットします．

b) 処理選択（"SLCT.BS"）

c)以下のプログラムを選択します．

c) マスタ管理（"MSTM.BS"）

品目マスタの照会，削除，登録，変更の各処理を行ないます．

d) 出庫先テーブル管理（"SKSM.BS"）

出庫先テーブルファイルの照会，削除，登録，変更を行ないます．

e) 入庫先テーブル管理（"NKSM.BS"）

入庫先テーブルファイルの照会，削除，登録，変更を行ないます．

f) 出庫処理（"SKPS.BS"）

出庫日，出庫先コード，伝票番号，品目コード，取引区分，出庫数量を入力して，品目マスタの更新，発注点切れチェック，品目別入出庫明細ファイル，出庫先別出庫明細ファイルの作成を行ないます．

g) 入庫処理（"NKPS.BS"）

入庫日，入庫先コード，伝票番号，品目コード，取引区分，入庫数量を入力して，品目マスタの更新・発注点切れチェック，品目別入出庫明細ファイル，入庫先別入庫明細ファイルの作成を行ないます．

h) 発注処理（"HCPS.BS"）

発注日，発注先コード，伝票番号，品目コード，発注数量を入力して，品目マスタの更新，発注点切れチェックを行ないます．

i) 作表処理（"SHYO.BS"）

在庫マスタ，出庫先テーブル，入庫先テーブル，発注点切れ品目の一覧表を作成します．さらに，

分類処理後，品目別入出庫明細表，出庫先別出庫明細表，入庫明細表を作成します．

j) 在庫調整（"ZKCM．BS"）

品目マスタの全品目について，現在残を前残に移し，入庫数および出庫数をクリアします．

k) ファルイニシャライズ（"FINT．BS"）

各マスタ，ファイルに登録されている全レコードを消去します．ファイル毎の消去および全ファイルの消去のいずれでも指定可能です．

13·3·3 ファイルデザイン

次に，品目マスタ，出庫先テーブル，出庫明細ファイルのファイルデザインを，代表として示します．

品　目　マ　ス　タ	"MST．SC"	1レコード 128バイト　2レコード／1セクタ
出庫先テーブルファイル	"STBL．SC"	1レコード　32バイト　8レコード／1セクタ
出庫明細ファイル	"SMS．SC"	1レコード　64バイト　4レコード／1セクタ

給与計算システムの場合は，1レコード／1セクタとして扱っていましたが，今の場合，上のように，1セクタに複数のレコードということで扱っています．フロッピーディスクの読み書き（GET／PUT）は，1セクタを対象としていますので，1セクタの中に複数のレコードの場合は，プログラム内で，そのように扱うくふうが必要です．

〔品目マスタ，"MST．SC"〕

登録情報	品目コード	品　目　名	入庫先コード	最低在庫数	前月末在庫数	当月入庫数	当月出庫数
I	I	C－10	I	D	D	D	D

現在庫数	発　注　残	在庫表示割	ブランク
D	D	I	C－62

	登録情報	品目コード	品　目　名	入庫先コード	最低在庫
	I	I	C－10	I	D

それぞれのプログラムの処理概要は，次の通りです．

a) 日時設定（"DTSET.BS"）

処理開始にあたって，処理日および開始時刻を初期セットします．

b) 処理選択（"SLCT.BS"）

c)以下のプログラムを選択します．

c) マスタ管理（"MSTM.BS"）

品目マスタの照会，削除，登録，変更の各処理を行ないます．

d) 出庫先テーブル管理（"SKSM.BS"）

出庫先テーブルファイルの照会，削除，登録，変更を行ないます．

e) 入庫先テーブル管理（"NKSM.BS"）

入庫先テーブルファイルの照会，削除，登録，変更を行ないます．

f) 出庫処理（"SKPS.BS"）

出庫日，出庫先コード，伝票番号，品目コード，取引区分，出庫数量を入力して，品目マスタの更新，発注点切れチェック，品目別入出庫明細ファイル，出庫先別出庫明細ファイルの作成を行ないます．

g) 入庫処理（"NKPS.BS"）

入庫日，入庫先コード，伝票番号，品目コード，取引区分，入庫数量を入力して，品目マスタの更新・発注点切れチェック，品目別入出庫明細ファイル，入庫先別入庫明細ファイルの作成を行ないます．

h) 発注処理（"HCPS.BS"）

発注日，発注先コード，伝票番号，品目コード，発注数量を入力して，品目マスタの更新，発注点切れチェックを行ないます．

i) 作表処理（"SHYO.BS"）

在庫マスタ，出庫先テーブル，入庫先テーブル，発注点切れ品目の一覧表を作成します．さらに，

分類処理後，品目別入出庫明細表，出庫先別出庫明細表，入庫明細表を作成します．

j) 在庫調整（"ZKCM.BS"）

品目マスタの全品目について，現在残を前残に移し，入庫数および出庫数をクリアします．

k) ファルイニシャライズ（"FINT.BS"）

各マスタ，ファイルに登録されている全レコードを消去します．ファイル毎の消去および全ファイルの消去のいずれでも指定可能です．

13·3·3 ファイルデザイン

次に，品目マスタ，出庫先テーブル，出庫明細ファイルのファイルデザインを，代表として示します．

品目マスタ	"MST.SC"	1レコード 128バイト 2レコード／1セクタ
出庫先テーブルファイル	"STBL.SC"	1レコード 32バイト 8レコード／1セクタ
出庫明細ファイル	"SMS.SC"	1レコード 64バイト 4レコード／1セクタ

給与計算システムの場合は，1レコード／1セクタとして扱っていましたが，今の場合，上のように，1セクタに複数のレコードということで扱っています．フロッピーディスクの読み書き（GET／PUT）は，1セクタを対象としていますので，1セクタの中に複数のレコードの場合は，プログラム内で，そのように扱うくふうが必要です．

〔品目マスタ，"MST.SC"〕

登録情報	品目コード	品目名	入庫先コード	最低在庫数	前月末在庫数	当月入庫数	当月出庫数
I	I	C－10	I	D	D	D	D

現在庫数	発注残	在庫表示割	ブランク
D	D	I	C－62

	登録情報	品目コード	品目名	入庫先コード	最低在庫
	I	I	C－10	I	D

〔出庫明細ファイル、"SMS. SC"〕

登録情報	取引コード	取引日	出庫先コード	出庫先名	伝票No.	行No.	品目コード	品目名	区分コード	取引数量
I	I	C－8	I	C－10	I	I	I	C－10	I	D

①

ブランク

C－14

②

③

④

13・4　統計計算プログラム

本節では基礎的な統計計算の方法について，簡単なプログラム例を掲げて解説します．統計的手法には実にさまざまなものがありますが，多かれ少なかれ数式を伴います．ですからその数式を誤解なく把握して，それをプログラムとして表現しなければなりません．

数式や計算手順をよく把握するには，プログラムを組む前に手計算を行なってみるのが有益です．本節の中でも手計算の過程を示してあります．

本節では統計処理における計算の部分を中心にして述べてありますが，さらに進んだ統計的な分析を行なうためには，データをディスクあるいはテープファイルに記録することも必要になると思います．それらについては本節では省略してあります．

13・4・1　平均値・標準偏差の計算

データの集まりを統計学的に処理する場合，平均値と標準偏差は最も基礎的な統計量です．本項ではこれらを計算するプログラムを示しますが，その前に用語と計算式について説明しておきます．

データを次のような記号で表わすことにします．n はデータの件数であり，x_i は i 番目のデータです．

$$x_1,\ x_2,\ x_3,\ \cdots\cdots,\ x_i,\ \cdots\cdots,\ x_n$$

平均値はデータの集団の中心的な位置を表わすものです．データの合計を件数で割れば平均値が求められます．平均値を $\overline{x}$ という記号で表わすと，次式のようになります．$\sum$ は「合計」を意味する記号です．

$$\overline{x}=\frac{1}{n}\sum_{i=1}^{n}x_i=\frac{1}{n}(x_1+x_2+x_3+\cdots\cdots+x_n)$$

標準偏差はデータの集団のバラツキの大きさを表わすものです．これを s という記号で表わすと，s は次式で定義されます．

$$s=\sqrt{\frac{\sum(x_i-\overline{x})^2}{n}}$$

コンピュータで計算する場合は，この定義式では不便なので，これを変形した次の式を用います．なぜ不便かというと，上式では $\overline{x}$（平均値）を求めた後で，もう一度 x_i（個々のデータ）を読まねばならないからです．

$$s=\sqrt{\frac{1}{n}\left(\sum x_i^2-(\sum x_i)^2/n\right)}$$

なお，標準偏差の式の中にある次のものを，x の偏差平方和と呼び，$S(xx)$ という記号で表わします．

$$S(xx)=\sum(x_i-\overline{x})^2=(x_1-\overline{x})^2+(x_2-\overline{x})^2+\cdots\cdots+(x_n-\overline{x})^2=\sum x_i^2-(\sum x_i)^2/n$$

また，偏差平方和を n で割ったものを分散と呼び，s^2 という記号で表わします．したがって次のような関係になります．

$$s=\sqrt{s^2}=\sqrt{\frac{S(xx)}{n}}$$

したがって，$\sum x_i, \sum x_i^2$ および n を求めれば，平均値や標準偏差を計算できるわけです．プログラムを作る前に，簡単なデータを使って手計算でやってみましょう．次のような計算表を使います．

i	x_i	x_i^2
1	3	3
2	7	49
3	6	36
4	1	1
5	3	9
合計	20	104

$n=5$　　$\sum x_i^2=104$　　$\sum x_i=20$

平均値：$\bar{x}=\frac{1}{n}\sum x_i=20/5=4$

偏差平方和：$S(xx)=\sum x_i^2-(\sum x_i)^2/n=104-20^2/5=24$

分散：$s^2=S(xx)/n=24/5=4.8$

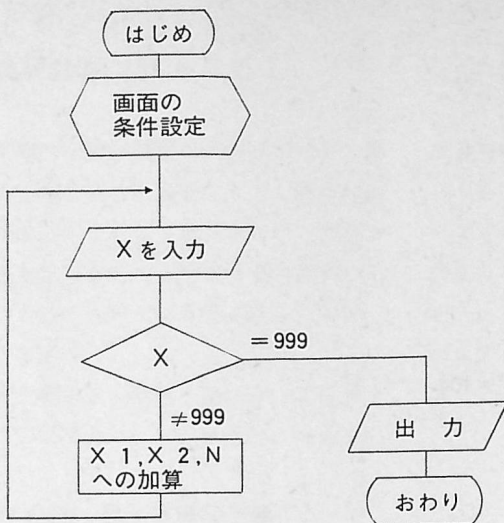
標準偏差：$s=\sqrt{s^2}=\sqrt{4.8}=2.19$

平均値および標準偏差の意味や計算式は以上のとおりです．次にプログラム例を掲げます．

このプログラムでは，データとして，学校でのテストの得点を対象としています．平均点と標準偏差の他に，最高点，最低点，合格者人数および合格率などを同時に求めるようにしてあります．テストの得点は0点から100点までの範囲内にあるものとします．60点以上の者を合格者とします．

最高点，最低点その他の計算はあとまわしにして，まず平均点と標準偏差を求める部分だけを示しましょう．

```
10  WIDTH 40, 25
20  CONSOLE 0, 25, 0, 0
50  PRINT CHR$ (12)
60  INPUT "テンスウ="; X
70  IF X=999 GOTO 200
100 X1=X1+X:X2=X2+X*X:N=N+1
150 GOTO 60
200 PRINT CHR$ (12)
220 PRINT "ヘイキンテン="; X1/N
230 PRINT "ヒョウジュンヘンサ="; SQR ((X2-X1*X1/N)/N)
900 END
```

上記のプログラムの流れ図は次のようになります．

データの終わりを示すために999点という値を用いています．終わりのしるしとしては，データとして存在しない値を用います．X 1，X 2，Nは，それぞれ$\sum x_i$，$\sum x_i^2$，人数を求めるための変数です．$\overline{x}$およびsの計算式はPRINT文の中に書いてあります．

標準偏差の計算というと何かむずかしそうに思えますが，実は上記のように非常に簡単にできるのです．要するに，手計算と同様なことをコンピュータにやらせればよいのです．

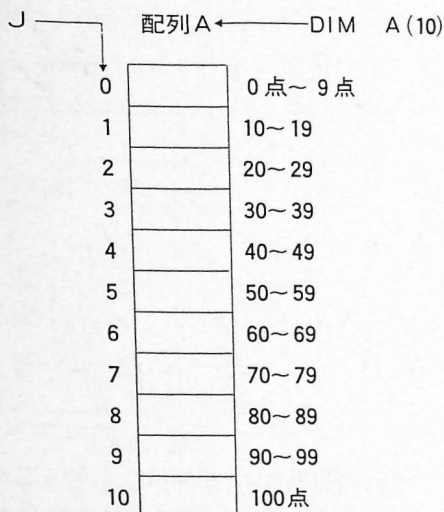
さて，最高点その他を求める手順を示しましょう．上記のプログラムに，次のものを追加すればよいのです．

```
40   X 0=100：X 9= 0
80   IF  X>= 0  AND  X=<100  GOTO  100
90   PRINT "ヤリナオシ！"：GOTO  60
120  IF  X>X 9  THEN  X 9=X
130  IF  X<X 0  THEN  X0=X
140  IF  X>=60  THEN  G=G+ 1
210  PRINT "ニンズウ="； N
240  PRINT "サイコウテン="； X 9
250  PRINT "サイテイテン="； X 0
260  PRINT "ゴウカクニンズウ="； G
270  PRINT "ゴウカクリツ="； G/N＊100 ； "%"
```

80番のIF文でデータのチェック（検査）をしています．0点～100点の範囲になければ，入力のやりなおしをさせます．X 0は最低点を求めるための変数であり，初期値として100点（最も大きな値）をセットしておいて，それより小さいデータがあったならば，それを記憶しておきます．X 9はその逆で，最高点を求めるための変数です．Gは合格人数を数えるための変数です．

13・4・2　得点のランク別集計

本項ではランク別集計の例として，学校でのテストの得点を対象としたプログラムを示します．問題状況の設定は前項のプログラムと同じものとします．

次図に示すように，0点～100点の得点範囲を11個のランクに分け，それぞれのランクに該当する人数を集計します．このために，プログラムの中でAという名前の配列を用います．

前項のプログラムに下記のものを追加してください．そうすれば，平均値や標準偏差などの計算と同時にランク別集計も行なうことができます．

```
30   DIM  A(10)
110  J=INT(X/10):A(J)=A(J)+1
300  PRINT:PRINT "ランク","ニンズウ"
310  FOR J=10 TO 0 STEP -1
320  PRINT J*10;"-",A(J)
330  NEXT J
```

110番の行で用いているINTは，小数部分を丸める関数です．例えば入力した点数（X）が47点であったとすれば，Jの値は次のようになります．

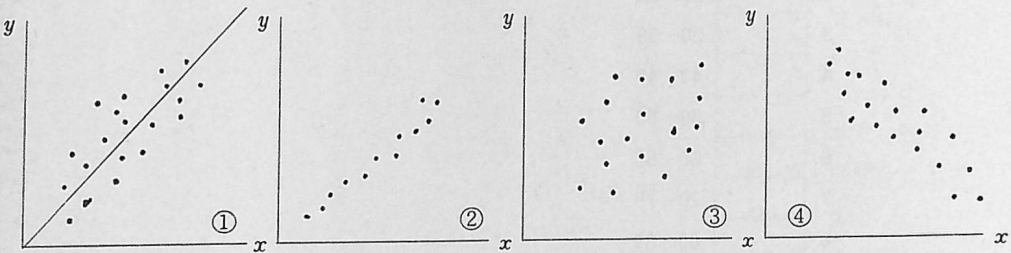
J＝INT(X／10)⟶INT(47／10)⟶INT(4.7)⟶4

そこで，A(J)すなわち配列の中のA(4)の場所に1が加算されることになります．A(4)は40点から49点までの得点者の人数を集計する場所です．このプログラムでは，ランクの幅が10点刻みなので，それを利用して配列内の位置(J)を求めているのです．

前項および本項に掲げたプログラムはテストの得点を対象としたものですが，その他の問題に対しても，部分的に修正を施せば適用できます．またランク別集計の結果を棒グラフ(ヒストグラム)として画面に表示すれば，さらに良いプログラムになります．各自で考えてみてください．

13·4·3　相関係数と回帰直線

例えば多数の人間について，身長と体重とを測定したとします．非常に太った人もやせた人もおりますが，全体的に見れば，身長が高い人ほど体重も重いという傾向があると思われます．このような場合，身長と体重とは相関関係があるといいます．

身長と体重に限らず，いろいろな事象において，相関関係が見られます．一般に x と y との相関関係を図示すると次のようになります．これを散布図といいます．

上図の①の程度の関係はよく見られます．②は非常に強い相関があります．③はほとんど相関がありません．④は逆向きの相関です．

相関関係の強さを表わす尺度として，相関係数が用いられます．相関係数の値は－１と＋１の間にあります．上図②のように相関が強い場合は，相関係数の値は１に近くなります．③のように相関が弱い場合は，相関係数は０に近くなります．④のように逆向きの場合は，相関係数の値はマイナスになります．

上図の①には直線が書いてありますが，これはデータ（点）の集まりに最も良く適合するように引いたものです．このような直線を回帰直線といいます．

相関係数を r という記号で表わすことにすると，r は次の式で求められます．

$$r=\frac{\mathrm{S}(xy)}{\sqrt{\mathrm{S}(xx)\cdot\mathrm{S}(yy)}}$$

ただし
$$\begin{cases}\mathrm{S}(xx)=\sum x_i^2-(\sum x_i)^2/n & \cdots\cdots\cdots x\text{の偏差平方和}\\ \mathrm{S}(yy)=\sum y_i^2-(\sum y_i)^2/n & \cdots\cdots\cdots y\text{の偏差平方和}\\ \mathrm{S}(xy)=\sum x_i y_i-(\sum x_i)(\sum y_i)/n & \cdots\cdots x\text{と}y\text{の偏差積和}\end{cases}$$

また，回帰直線の方程式は次のように求められます．

$$y=a+bx \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\ x\text{に対する}y\text{の回帰直線}$$

ただし
$$\begin{cases}b=\dfrac{\mathrm{S}(xy)}{\mathrm{S}(xx)}\\ a=(\sum y_i-b\sum x_i)/n\end{cases}$$

プログラムを作る前に，簡単なデータを使って手計算でやってみましょう．次のような計算表を使います．見てわかるように，S (xx)の計算の部分は，標準偏差の場合と同じです．

i	x_i	y_i	x_i^2	y_i^2	$x_i y_i$
1	3	2	9	4	6
2	7	4	49	16	28
3	6	5	36	25	30
4	1	3	1	9	3
5	3	1	9	1	3
合計	20	15	104	55	70

$n = 5$

$\sum x_i = 20$

$\sum y_i = 15$

$\sum x_i^2 = 104$

$\sum y_i^2 = 55$

$\sum x_i y_i = 70$

$$\mathrm{S}(xx) = \sum x_i^2 - (\sum x_i)^2/n = 104 - 20\times 20/5 = 24$$

$$\mathrm{S}(yy) = \sum y_i^2 - (\sum y_i)^2/n = 55 - 15\times 15/5 = 10$$

$$\mathrm{S}(xy) = \sum x_i y_i - (\sum x_i)(\sum y_i)/n = 70 - 20\times 15/5 = 10$$

$$r = \frac{10}{\sqrt{24\times 10}} = 0.6455$$ ………相関係数

$$b = \mathrm{S}(xy)/\mathrm{S}(xx) = 10/24 = 0.4167$$

$$a = (\sum y_i - b\sum x_i)/n = (15 - 0.4167\times 20)/5 = 1.333$$

$$y = 1.333 + 0.4167x$$ ………回帰直線

プログラムは次のようになります．

```
10   WIDTH 40, 25
20   CONSOLE 0, 25, 0, 0
30   PRINT CHR$(12)
40   INPUT "データ(x, y)="; X, Y
50   IF X=999 AND Y=999 GOTO 100
60   X 1=X 1+X : X 2=X 2+X*X
70   Y 1=Y 1+Y : Y 2=Y 2+Y*Y
80   XY=XY+X*Y : N=N+ 1
90   GOTO 40
100  S 1=XY-X 1*Y 1/N
110  S 2=X 2-X 1*X 1/N
120  S 3=Y 2-Y 1*Y 1/N
130  R=S 1/SQR(S 2*S 3)
140  B=S 1/S 2 : A=(Y 1-B*X 1)/N
150  PRINT CHR$(12)
```

```
160 PRINT "データ　ケンスウ：n="；N
170 PRINT：PRINT "ソウカン　ケイスウ：r="；R
180 PRINT：PRINT "カイキ　チョクセン：y=a +bx "
190 PRINT：PRINT "　a="；A
200 PRINT：PRINT "　b="；B
210 BEEP：END
```

このプログラムでは，入力したXおよびYがともに999のとき，データの終わりとみなしています．計算の部分は平均値や標準偏差のプログラムとよく似ているので，手計算の例とも照らし合わせてみれば理解できると思います．

また，散布図や回帰直線をディスプレイ画面にグラフィック表示すると，さらによいと思います．ちょっと面倒かも知れませんが，興味のある人は挑戦してみてください．

〔付録〕 エラーメッセージ

ERR コード	エラーメッセージ	意味
1	NEXT without FOR	FOR－NEXTが正しく対応していない．（NEXTが多すぎる）
2	Syntax error	文法がまちがっている．プログラム中に規定のステートメント以外のものがある．
3	RETURN without GOSUB	GOSUB－RETURNが正しく対応していない．（RETURNだけがある）
4	Out of Data	READ文で読まれるべきデータがDATA文の中に用意されていない．
5	Illegal function call	ステートメントの機能の呼び方がまちがっている．
6	Over flow	入力された数値や演算結果が許容される範囲を外れている．
7	Out of memory	メモリ容量が足りなくなった．（プログラムが長すぎる，配列が大きすぎる等）
8	Undefined line number	必要とされるプログラム行（GOTOの飛び先など）が定義されていない．
9	Subscript out of range	配列変数の添字が規定の範囲内にない．
10	Redimensioned array	同じ配列を再定義している．（同じDIM文が複数回実行された．）
11	Division by zero	0による割算が実行された．
12	Illegal direct	ダイレクト・ステートメントとして使えないコマンドが入力された．
13	Type mismatch	代入文などで式の左右の型が一致してい

		ない．（数値とストリングなど）
14	Out of string space	ＣＬＥＡＲ文などで指定した，ストリング変数用メモリエリアが足りなくなった．
15	String too long	ストリング（引用符で囲まれた文字列）が長すぎる．（255文字をこえた）
16	String formura too complex	文字式が複雑すぎる．（カッコのネスティングレベルが多すぎる等）
17	Can't continue	ＣＯＮＴコマンドを入力しても続行できない．（ポインタが破壊されている）
18	Undefined user function	参照されたユーザー関数の定義（ＤＥＦ文による）がなされていない．
19	No RESUME	エラー処理後，プログラムの実行を再開することができない．
20	RESUME without error	エラーとＲＥＳＵＭＥが対応していない．（エラーがないのにＲＥＳＵＭＥした．）
21	Unprintable error	メッセージの定義されていないエラー．
22	Missing operand	ステートメント中に必要とされるパラメータなどが指定されていない．
23	Line buffer overflow	ＢＡＳＩＣが，行を入力する際，一行の長さが有効範囲を越えている．
24	Position not on Screen	指定したカーソル位置などが，画面の範囲外になっている．
25	Bad File Data	ファイル上にあるデータの形式がまちがっている．
26	Disk BASIC Feature	ディスクが接続されていないとき，ディスクＢＡＳＩＣの命令を実行した．
27	Communications Buffer overflow	周辺機器との入出力のためのバッファがオーバーフローした．
28	Port not initialized	インタフェイス用のＬＳＩの機能設定がなされていない．
29	Tape read ERROR	カセットからの入力が正しく行なわれていない．（テープの読み誤まりなど）

ディスクＢＡＳＩＣの場合のエラー（ディスクのない場合はエラー26, 21となる）

50	Field overflow	ＦＩＥＬＤ文で256バイトをこえる文字を割り当てた．
51	Internal error	ＢＡＳＩＣ内部でのエラー．
52	Bad file number	オープンされていないファイル・ナンバーをアクセスした．
53	File not found	ＬＯＡＤ，ＫＩＬＬ，ＯＰＥＮなどで，存在しないファイルをアクセスした．
54	Bad file mode	シーケンシャル・ファイルでＯＰＥＮしたファイルに対して，ランダム・アクセスをした．またはその逆．
55	File already open	すでに開かれているファイルに対してＯＰＥＮやＫＩＬＬを実行した．
57	Disk I/O error	ディスクにリード・ライトエラーが生じ読み直しても修正できなくなった．
58	File already exists	ＮＡＭＥ文によって定義されたファイルネームが，すでに登録されている．
61	Disk Full	ディスク上のすべての場所を使い切った．
62	Input past end	ファイルのすべてのデータを読み尽した後に，ＩＮＰＵＴ文が実行された．
63	Bad record number	ＰＵＴ，ＧＥＴ文で，レコード・ナンバーが0～32767の範囲にない．
64	Bad file name	不適当なファイルネームを使った．
66	Direct statement in file	アスキー・フォーマットでのＬＯＡＤ中にダイレクト・ステートメントがあった．
67	Too many files	256種類以上のファイルを作り出した．

〔索　引〕

〔ア〕

〔カ〕

〔サ〕

プログラムを組む人のための

NEC-PC8001

N-BASICプログラミング教本

著　者　横山　淳
発行者　川口勝治
発行所　廣済堂出版

〒105
東京都港区芝2-23-13
電話03-453-1201(代)
振替　東京8　164137番

印刷所　廣済堂印刷

定価は、カバーに明示してあります。
落丁・乱丁本はお取替えいたします。